

Una veloce introduzione al Linguaggio C++

https://github.com/squillero/introduzione_al_cpp

Giovanni Squillero
squillero@polito.it



1

Copyright © 2019 by Giovanni Squillero

- Permission to make digital or hard copies for personal or classroom use of this file is granted without fee provided that copies are not distributed for profit and that copies preserve both the copyright notice and the full reference to the source repository. To republish, to redistribute to lists, or to post on servers, contact the Author.
- This file is offered as-is, without any warranty.



3

Argomenti

- Fondamenti del C++
- Classi e Oggetti
- Memoria dinamica e puntatori
- Template e libreria standard

Sistemi Embedded



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

5

5

Hello world!



```
// Il mio primo programma in C++  
  
#include <iostream>  
int main() {  
    std::cout << "Il corso è iniziato!" << std::endl;  
    return 0;  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

6

6

Cosa è successo?

- File sorgente [.cpp .cc .cxx .C]
- File include [.h ...]
- Compilatore
- File oggetto [.o .obj]
- Linker
- Eseguibile [.exe a.out foo]

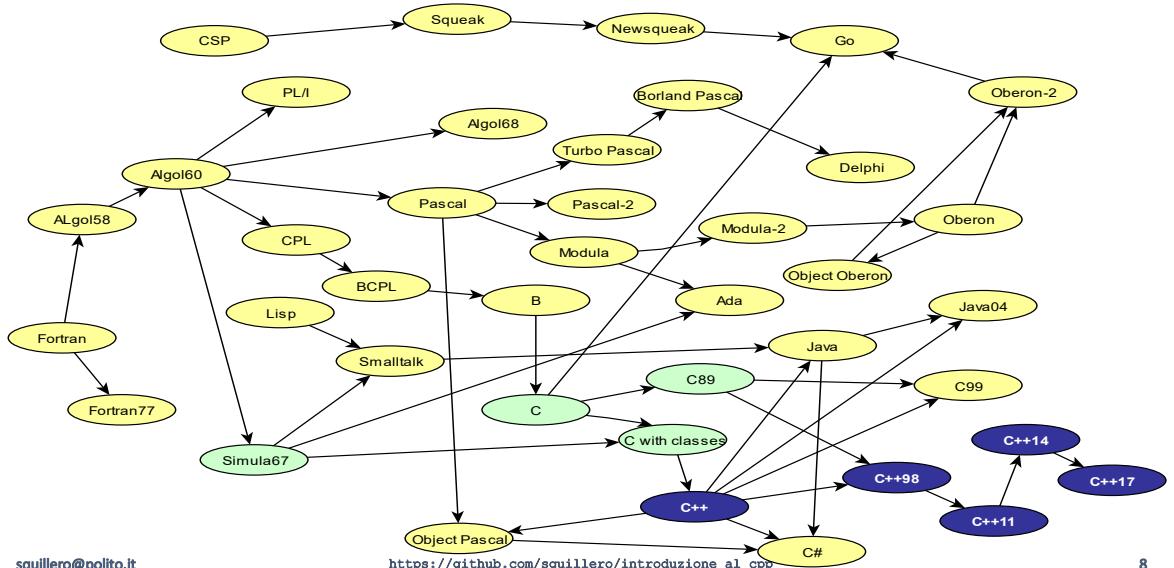
squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

7

7

Breve Storia del C++



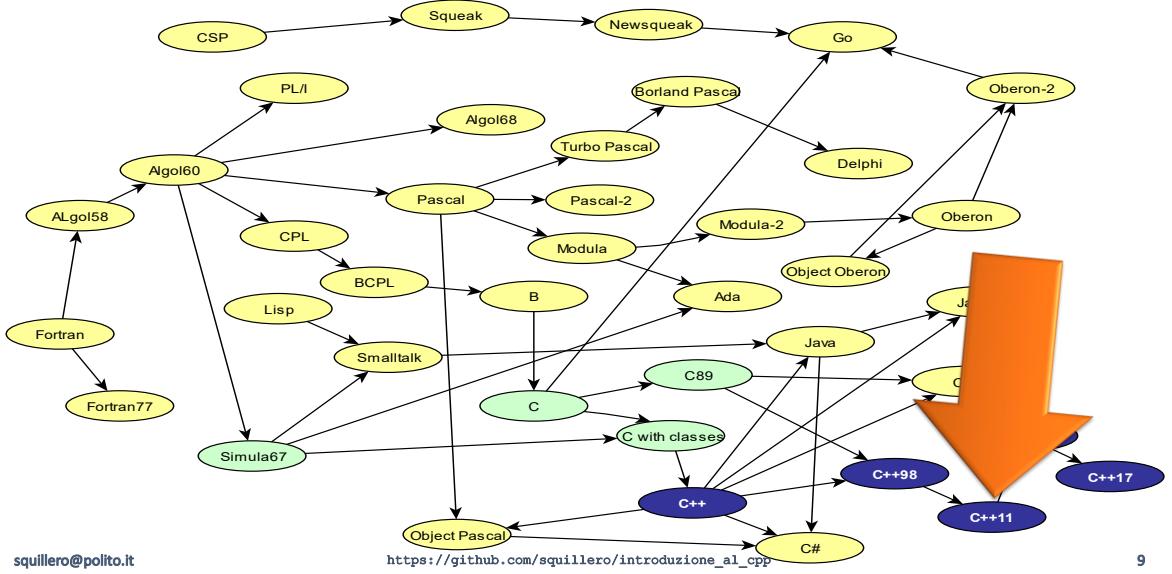
squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

8

8

Breve Storia del C++



squillero@polito.it

9

9

Breve Storia del C++

- 1979: *C-with-classes* di Bjarne Stroustrup
 - 1983: Il “C con classi” diventa C++
 - 1985: *C++ Programming Language*, 1° edizione
 - 1989-1999: Prima standardizzazione ANSI/ISO C++98
 - 2011: ANSI/ISO C++11 (anche noto come C++0x)
 - 2014: ANSI/ISO C++14
 - 2017: ANSI/ISO C++17
 - ...

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

10

10



C++

- Compiled vs. Interpreted (vs. JIT-compiled)
- High level vs. Low level
- Type system
 - Type Strength: Strong or Weak
 - Type Expression: Manifest or Inferred
 - Type Checking: Static or Dynamic
 - Type Safety: Safe or Unsafe



C++

- Compiled vs. Interpreted (vs. JIT compiled)
- High level vs. Low level
- Type system
 - Type Strength: Strong or Weak
 - Type Expression: Manifest or Inferred
 - Type Checking: Static or Dynamic
 - Type Safety: Safe or Unsafe

<http://cppreference.com/>

The screenshot shows the homepage of cppreference.com. At the top, there's a navigation bar with tabs for "Page" and "Discussion". To the right of the navigation bar are links for "View", "View source", "History", and "Actions". The main content area has a header "C++ reference" with a subtitle "C++98, C++03, C++11, C++14, C++17, C++20". Below the header is a large sidebar containing links to various parts of the standard library:

- Language**
 - Compiler support
 - Basic concepts
 - C++ Keywords
 - Preprocessor
 - Expressions
 - Declaration
 - Initialization
 - Functions
 - Statements
 - Classes
 - Templates
 - Exceptions
- Headers – Library concepts**
- Language support library**
 - Type support – traits (C++11)
 - Program utilities
 - Relational comparators (C++20)
 - numeric_limits – type_info
 - initializer_list (C++11)
- Diagnostics library**
- General utilities library**
 - Smart pointers and allocators
 - Date and time
 - Function objects – hash (C++11)
 - String conversions (C++17)
 - Utility functions
 - pair – tuple (C++11)
 - optional (C++17) – any (C++17)
 - variant (C++17)
- Containers library**
 - Null-terminated strings
 - byte – multibyte – wide
 - Sequence containers
 - Associative containers
 - Unordered associative containers
 - Container adaptors
- Strings library**
 - basic_string
 - basic_string_view (C++17)
- Input/output library**
 - Stream-based I/O
 - Synchronized output (C++20)
 - I/O manipulators
- Numerics library**
 - Complex numbers
 - Special math functions (C++17)
 - Numeric algorithms
 - Pseudo-random number generation
 - Floating-point environment (C++11)
 - complex – valarray
- Localizations library**
 - Regular expressions library (C++11)
 - Atomic operations library (C++11)
 - Thread support library (C++11)
- Filesystem library (C++17)**

At the bottom of the sidebar, there are sections for "Technical specifications" (Standard library extensions, Standard library extensions V2, Parallelism library extensions, Concurrency library extensions, Concepts, Ranges, Transactional Memory, Feature Test Recommendations), and a footer with the email address "squillero@polito.it".

13

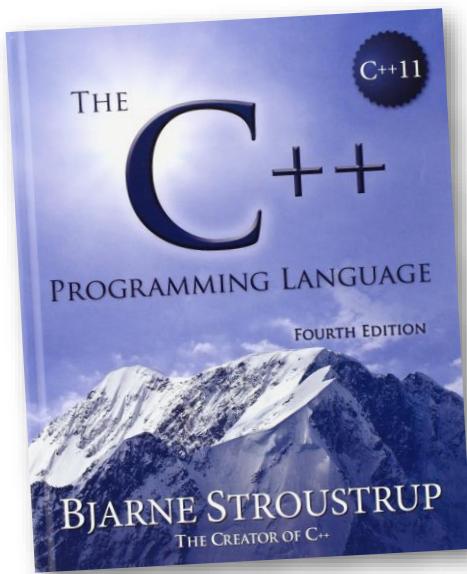
<http://cppreference.com/>

This screenshot shows the C++ reference site with a DuckDuckGo search overlay. The search bar at the top contains the query "!cpp". Below the search bar, a dropdown menu lists several search results:

- !cppreference....
- !cpp
- !cpr
- !cppref...
- C++Samples
- !cppsamples

A button labeled "More about !bangs" is visible. The main content area of the page is partially obscured by the search interface but shows the same sidebar and footer as the previous screenshot.

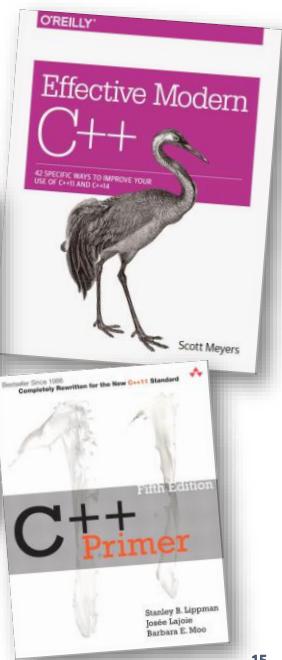
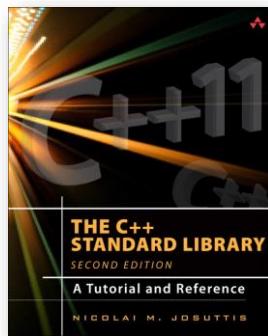
14



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

Libri



15

15

Zeal: Offline documentation browser

8.11. pprint — Data pretty printer - Zeal

File Edit Tools Help

python2 pprint

pprint pprint

PRINT_EXPR

PyErr_Print

PyErr_PrintEx

PyOS_snprintf

cgi.print_form

pprint.pformat

PyObject_Print

DYNAMIC_VPRINTF

pprint.PrettyPrinter

pprint.pformat

pprint.isrecursive

pprint.saferepr

pprint.isreadable

pprint.print

pprint.PrettyPrinter.pprint

pprint.PrettyPrinter.format

pprint.PrettyPrinter.isreadable

pprint.PrettyPrinter.pformat

pprint.PrettyPrinter.isrecursive

8.11. pprint — Data pretty printer - Zeal

std::printf, std::... CMAKE_INSTALL_...

pprint.**printf**(object, stream=None, indent=1, width=80, depth=None, *, compact=False)

Prints the formatted representation of object on stream, followed by a newline. If stream is None, sys.stdout is used. This may be used in the interactive interpreter instead of the `print()` function for inspecting values (you can even reassign `print` = `pprint.pprint` for use within a scope). `indent`, `width`, `depth` and `compact` will be passed to the `PrettyPrinter` constructor as formatting parameters.

Changed in version 3.4: Added the `compact` parameter.

```
>>> import pprint
>>> stuff = ['spam', 'eggs', 'lumberjack', 'knights', 'ni']
>>> stuff.insert(0, stuff)
>>> pprint.pprint(stuff)
[Recursion on list with id=...>,
 'spam',
 'eggs',
 'lumberjack',
 'knights',
 'ni']
```

<https://zealdocs.org/>

https://github.com/squillero/introduzione_al_cpp

16

Perché C++ oggi?

- Molto controllo
- Programmazione procedurale
- Programmazione orientata agli oggetti
- Programmazione «generica»
- Troppo complesso
- Tante caratteristiche, fra loro incompatibili

squillero@polito.it

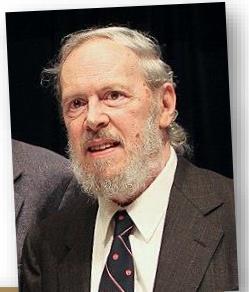
https://github.com/squillero/introduzione_al_cpp

17

17

Perché C++ oggi?

- C++ vs. C
- C++ vs. Java
- C++ vs. Python
- C++ vs. Go



18

Introduzione al C++

«Se un linguaggio di programmazione non fa cambiare il vostro modo di pensare, non vale la pena impararlo»

— Bjarne Stroustrup,
Programming: Principles and Practice Using C++



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

19

Fondamenti

Una veloce introduzione al C++

Giovanni Squillero
squillero@polito.it

Hello world!



```
// Il mio primo programma in C++  
  
#include <iostream>  
  
int main() {  
    std::cout << "Il corso è iniziato!" << std::endl;  
    return 0;  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

21

21

Elementi di un programma C++

- // Commenti
- #direttive per il pre-processore
- { Blocchi }
- Funzioni()
- Istruzioni, operatori
- Parole riservate (keywords)
- Name::space

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

22

22

Keywords

```
alignas, alignof, and, and_eq, asm, auto, bitand, bitor,  
bool, break, case, catch, char, char16_t, char32_t, class,  
compl, const, constexpr, const_cast, continue, decltype,  
default, delete, do, double, dynamic_cast, else, enum,  
explicit, export, extern, false, float, for, friend, goto,  
if, inline, int, long, mutable, namespace, new, noexcept,  
not, not_eq, nullptr, operator, or, or_eq, private,  
protected, public, register, reinterpret_cast, return,  
short, signed, sizeof, static, static_assert, static_cast,  
struct, switch, template, this, thread_local, throw, true,  
try, typedef, typeid, typename, union, unsigned, using,  
virtual, void, volatile, wchar_t, while, xor, xor_eq,
```

Tipi e Oggetti

- **Tipo**
 - Definisce l'insieme dei possibili valori e delle possibili operazioni
- **Oggetto**
 - Della memoria che contiene un *valore* di un certo *tipo*
- **Valore**
 - Un insieme di bit che sono interpretati in base al *tipo*
- **Variabile**
 - Un oggetto a cui è stato dato un nome

Tipi e Oggetti

- Dichiarazione
 - Una istruzione che assegna un nome ad un oggetto
- Definizione
 - Una istruzione che definisce uno spazio di memoria dove memorizzare un oggetto (e lo dichiara)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

25

25

Tipi e Oggetti

- Scope
 - Dove è possibile utilizzare un nome (nel codice sorgente)
- Lifespan
 - Quando è possibile utilizzare un oggetto
- Storage class
 - Dove viene memorizzato l'oggetto (in memoria)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

26

26

Nomi



- Usate nomi lunghi e descrittivi per variabili globali o «importanti»
- Usate nomi corti per variabili locali e con vita breve
- Possibili convenzioni
 - GlobaleCamelCase
 - Globale_importante
 - Globale_Importante
 - localeCamelCase
 - locale_ma_importante
 - ...



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

27



28

<https://google.github.io/styleguide/cppguide.html>

The screenshot shows the 'Table of Contents' section of the Google C++ Style Guide. The page has a header 'Google C++ Style Guide'. Below it is a table of contents organized into sections: 'C++ Version', 'Header Files', 'Scoping', 'Classes', and 'Functions'. Each section contains several sub-links. At the bottom of the page is a GitHub URL: https://github.com/squillero/introduzione_al_cpp.

C++ Version	
Header Files	Self-contained Headers The #define Guard Forward Declarations Inline Functions Names and Order of Includes
Scoping	Namespaces Unnamed Namespaces and Static Variables Nonmember, Static Member, and Global Functions Local Variables Static and Global Variables thread local Variables
Classes	Doing Work in Constructors Implicit Conversions Copyable and Movable Types Structs vs. Classes Inheritance Operator Overloading Access Control Declaration Order
Functions	Output Parameters Write Short Functions Reference Arguments Function Overloading Default Arguments Trailing Return Type Syntax

29

29

Tipi di dato fondamentali

- void
- bool
- caratteri
 - char
 - signed/unsigned char
 - wchar_t
 - char16_t
 - char32_t

30

Tipi di dato fondamentali: Interi

- Tipo di base: int
- Con/senza segno: signed/unsigned
- Dimensione:
 - short int (almeno 16 bit)
 - int (almeno 16 bit)
 - long int (almeno 32 bit)
 - long long int (almeno 64 bit)

Tipi di dato fondamentali: Virgola Mobile

- Singola precisione (float)
 - IEEE-754 32 bit
- Doppia precisione (double)
 - IEEE-754 64 bit
- Precisione estesa (long double)
 - Non IEEE-754, solitamente 80 bit

sizeof ()

```
int main() {  
    std::cout << "char: " << sizeof (char) << std::endl;  
  
    std::cout << "short int: " << sizeof (short int) << std::endl;  
    std::cout << "int: " << sizeof (int) << std::endl;  
    std::cout << "long int: " << sizeof (long int) << std::endl;  
    std::cout << "long long int: " << sizeof (long long int) << std::endl;  
  
    std::cout << "float: " << sizeof (float) << std::endl;  
    std::cout << "double: " << sizeof (double) << std::endl;  
    std::cout << "long double: " << sizeof (long double) << std::endl;  
}
```

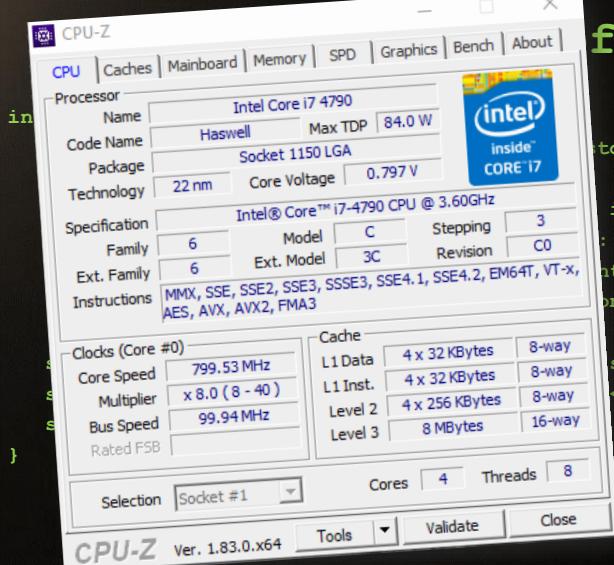
squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

33

f ()

```
std::cout << sizeof (char) << std::endl;  
std::cout << sizeof (short int) << std::endl;  
std::cout << sizeof (int) << std::endl;  
std::cout << sizeof (long int) << std::endl;  
std::cout << sizeof (long long int) << std::endl;  
  
std::cout << sizeof (float) << std::endl;  
std::cout << sizeof (double) << std::endl;  
std::cout << sizeof (long double) << std::endl;
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

34

Interi di dimensione fissa

- Un intero con/senza segno di esattamente x bit
 - int8_t, int16_t, int32_t, int64_t
 - uint8_t, uint16_t, uint32_t, uint64_t
- Il più veloce intero con/senza segno di almeno x bit
 - int_fast8_t, int_fast16_t, int_fast32_t, int_fast64_t
 - uint_fast8_t, uint_fast16_t, uint_fast32_t, uint_fast64_t
- Il più piccolo intero con/senza segno di almeno x bit
 - int_least8_t, int_least16_t, int_least32_t, int_least64_t
 - uint_least8_t, uint_least16_t, uint_least32_t, uint_least64_t

sizeof ()



```
int main() {
    std::cout << "int8_t: " << sizeof (int8_t) << std::endl;
    std::cout << "int16_t: " << sizeof (int16_t) << std::endl;
    std::cout << "int32_t: " << sizeof (int32_t) << std::endl;
    std::cout << "int64_t: " << sizeof (int64_t) << std::endl;
    std::cout << "int_fast8_t: " << sizeof (int_fast8_t) << std::endl;
    std::cout << "int_fast16_t: " << sizeof (int_fast16_t) << std::endl;
    std::cout << "int_fast32_t: " << sizeof (int_fast32_t) << std::endl;
    std::cout << "int_fast64_t: " << sizeof (int_fast64_t) << std::endl;
    std::cout << "int_least8_t: " << sizeof (int_least8_t) << std::endl;
    std::cout << "int_least16_t: " << sizeof (int_least16_t) << std::endl;
    std::cout << "int_least32_t: " << sizeof (int_least32_t) << std::endl;
    std::cout << "int_least64_t: " << sizeof (int_least64_t) << std::endl;
}
```

Range di valori

- Caratteri
 - char: -128 – 127 o 0 – 255
 - char16_t: 0 – 65,535
 - char32_t: 0 – 1,114,111

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

37

37

Range di valori

- Interi
 - 16 bit: 0 – 65,535
 - 32 bit: 0 – 4,294,967,295
 - 64 bit: 0 – 18,446,744,073,709,551,615
- Virgola mobile
 - Float: $\pm 3.4 \cdot 10^{\pm 38}$ con circa 7 cifre significative
 - Double: $\pm 1.7 \cdot 10^{\pm 308}$ con circa 15 cifre significative

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

38

38

Stringhe

- `std::string` → `std::basic_string<char>`
- `std::wstring` → `std::basic_string<wchar_t>`
- `std::u16string` → `std::basic_string<char16_t>`
- `std::u32string` → `std::basic_string<char32_t>`

Operazioni ed operatori

- Assegnamento (=)
- Operazione e assegnazione (★=)

Operazioni ed operatori (numeri)

- Assegnamento (=)
- Addizione (+), sottrazione (-), moltiplicazione(*), divisione (/)
- Resto della divisione o Modulo (%)
- Incremento (++) e decremento (--) di 1

Operazioni ed operatori (interi)

- Not (~)
- And (&)
- Or (|)
- Xor (^)
- Shift a destra (>>) e a sinistra (<<)

Operazioni ed operatori (stringhe)

- Concatenazione di stringhe (+)
- Aggiungi alla fine (+=)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

43

43

Operazioni di input/output

- Leggi da s in x (`s >> x`)
- Scrivi x in s (`s << x`)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

44

44

Confronti

- Uguale (==)
- Diverso (!=)
- Maggiore (>), maggiore o uguale (>=)
- Minore (<), minore o uguale (<=)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

45

45

Operazioni ed operatori (bool)

- Not (!)
- And (&&)
- Or (||)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

46

46

Variabili «auto»

- Il compilatore è capace di dedurre il tipo di una espressione
- Il tipo della variabile è dedotto dall'inizializzazione

```
auto x = 23;  
auto y = 2.3;
```

Inizializzazione

- int v1 = 7;
- int v2(7);
- int v3 = { 7 }; // strano, ma funziona sia in C sia in C++
- int v4 {7}; // nuovo in C++11
- int x1; // da evitare: x1 = 0 se non è una variabile locale
- int x2(); // oops!? è un prototipo!
- int x3 = {};// nuovo in C++11: x3 = 0 (valore di default)
- int x4{}; // nuovo in C++11: x4 = 0 (valore di default)
- auto a1 = int{7} // nuovo in C++11
- auto a = 7 // nuovo in C++11
- auto a = { 7 } // nuovo in C++11

Inizializzazione



- int v1 = 7;
- int v2(7);
- int v3 = { 7 }; // strano, ma funziona sia in C sia in C++
- int v4 {7}; // nuovo in C++11
- int x1; // da evitare: x1 = 0 se non è una variabile locale
- int x2(); // oops!? è un prototipo!

La «list initialization» impedisce le «narrowing conversion»

unsigned int ok = -1;
unsigned int error {-1};

Biognerebbe usarla sempre, a meno che non ci sia una ottima ragione per fare diversamente...

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

49

49

Inizializzazione



- int v1 = 7;
- int v2(7);
- int v3 = { 7 }; // strano, ma funziona sia in C sia in C++
- int v4 {7}; // nuovo in C++11
- int x1; // da evitare: x1 = 0 se non è una variabile locale
- int x2(); // oops!? è un prototipo!
- int x3 = {};// nuovo in C++11: x3 = 0 (valore di default)
- int x4{}; // nuovo in C++11: x4 = 0 (valore di default)
- auto a1 = int{7} // nuovo in C++11
- auto a = 7 // nuovo in C++11
- auto a = { 7 } // nuovo in C++11

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

50

50

Inizializzazione



- int v1 = 7;
- int v2(7);
- int v3 = { 7 }; // strano, ma funziona sia in C sia in C++
- int v4 {7}; // nuovo in C++11
- int x1; // da evitare: x1 = 0 se non è una variabile locale
- int x2(); // oops!? è un prototipo!

Ok: è comunque un tipo di inizializzazione in C++11: x3 = 0 (valore di default)

«list initialization»

- auto a1 = int{7} // nuovo in C++11
- auto a = 7 // nuovo in C++11
- auto a = { 7 } // nuovo in C++11



51

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

51

Conversioni (Casting)



```
int foo = 4.2;           // Troncamento implicito
int bar = int(4.2);      // Troncamento esplicito
int baz = static_cast<int>(4.2); // Troncamento MOLTO esplicito

int gargle{static_cast<int>(4.2)}; // D'ho!?
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

52

52

Esercizio

- Dichiare 2 variabili stringa *nome* e *cognome*, inizializzate
 - Costruire una variabile stringa *nome_completo* concatenando nome e cognome (usare gli spazi che servono)
 - Dichiare una variabile intera: età
 - Dichiare una variabile double: peso
 - Stampare il tutto su std::cout
- Mi chiamo Bond, James Bond. Ho 42 anni, peso 79.8 kg.**
- Notez bien: mentire è lecito

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

53

53

Tipi di dato aggregati: struct

```
struct Agente {  
    std::string nome;  
    std::string cognome;  
    int anni;  
    double peso;  
};
```

```
int main() {  
    Agente bond;  
    bond.nome = "James";  
    ...  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

54

54

Tipi di dato aggregati: struct

- Inizializzazione

```
struct Agente {  
    std::string nome = "";  
    std::string cognome;  
    int anni = 0;  
    double peso;  
};
```

Tipi di dato aggregati: bit field

```
// «Solitamente» occupa 2 byte  
  
struct control_register {  
    // 3 bits: value of b1  
    // 2 bits: unused  
    // 6 bits: value of b2  
    // 2 bits: value of b3  
    // 3 bits: unused  
    unsigned char b1 : 3, : 2, b2 : 6, b3 : 2;  
};
```

Tipi di dato aggregati: union

```
union Agente2 {  
    int anni;  
    double peso;  
};
```

```
int main() {  
    Agente2 bob;  
    bob.anni = 23;  
    std::cout << bob.anni << std::endl;  
    bob.peso = 112.3;  
    std::cout << bob.anni << std::endl;  
}
```

Puntatori

- Una variabile può contenere l'indirizzo di un'altra variabile
 - Modificatore di dichiarazione *
 - Operatore &
 - Operatore *
- Portabilità
- **(void *) 0 vs. NULL vs. nullptr**

Riferimenti

- Una variabile può contenere un *riferimento* ad un'altra variabile
 - Modificatore di dichiarazione &
- Leggibilità/manutenibilità

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

59

59

Puntatori e Riferimenti



```
int foo = 42;
std::cout << foo << std::endl;           // 42

int& bar = foo;
++bar;
std::cout << foo << std::endl;           // 43

int* baz = &foo;
***baz;
std::cout << foo << std::endl;           // 44
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

60

60

Puntatori e Riferimenti



```
int foo = 42;
std::cout << foo << std::endl;      // 42
 $\downarrow$ 
int& bar = foo;
++bar;
std::cout << foo << std::endl;      // 43
 $\downarrow$ 
int* baz = &foo;
***baz;
std::cout << foo << std::endl;      // 44
 $\uparrow$ 
```

Modificatore di dichiarazione

Operatore

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

61

Puntatori

- Di che tipo sono *p1* e *p2*?

```
double* p1, p2;
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

62

Casting

- `reinterpret_cast`
 - Converte fra tipi di dato *reinterpretando* il pattern di bit

```
long long int i = 42;
double* p;

p = reinterpret_cast<double*>(&i);
std::cout << "i: " << &i << " -> " << i << std::endl;
std::cout << "p: " << p << " -> " << *p << std::endl;
```

```
i: 0x61ff00 -> 42
p: 0x61ff00 -> 2.07508e-322
```

Controllo di flusso

- While
- Do { } while
- For
- Range loop

i/o



```
int main() {
    std::string parola;

    // NB: EOF è CTRL-D in molti unix, CTRL-Z in windows
    while(std::cin >> parola) {
        std::cout << "Bravo, hai detto \"" << parola << "\"" << std::endl;
    }
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

65

65

Selezione

- if-else
- if-else-if
- switch

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

66

66

Esercizio

- Modificare il programma in modo che riconosca sequenze di parole uguali

```
Ciao
Bravo, hai detto "Ciao"
Mamma
Bravo, hai detto "Mamma"
Mamma
Bravo, hai detto "Mamma" di nuovo (2 volte)
Mamma
Bravo, hai detto "Mamma" di nuovo (3 volte)
Ciao
Bravo, hai detto "Ciao"
```

Esercizio

- Utilizzando if, while, e for, stampare pattern *simili* a questi

```
#####
##      ##
##      ##
##      ##
#####
```

```
#####
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
#####
```

Type-safety

- Un programma è «type-safe» quando utilizza gli oggetti in modo conforme con il loro tipo
- Conversioni sicure e non-sicure
 - «Narrowing conversion»

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

69

69

Funzioni

- Ricordate il prototipo
- NB: main è una funzione!

```
double avg(double a, double b) {  
    double avg = (a + b) / 2.0;  
    return avg;  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

70

70

Funzioni (passaggio di parametri)

- By value
- By reference (&)
- Puntatori (*)
- «Variadic functions»
- Valori di default dei parametri (nel prototipo)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

71

71

Variabili «costanti»

- Keyword: const

```
// nessuno modificherà mai foo
const int foo = 23;

// bar restituisce un riferimento ad qualcosa che nessuno deve modificare
const C& bar();

// baz non modificherà il suo parametro
void baz(const C& p);
```

- La variabile diventa «read-only»
- Il compilatore riesce ad ottimizzare meglio il codice

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

72

72

Puntatori «costanti»

- Keyword: const

```
int ci = 42;

// il dato puntato da p1 non è modificabile (NO: *p1 = nuovo_valore),
// ma p1 può cambiare (OK: p1 = &altra_variabile)
const int* p1 = &ci;
int const * p2 = &ci;

// il dato puntato da p3 è modificabile (OK: *p3 = nuovo_valore),
// ma p3 non può cambiare (NO: p3 = &altra_variabile)
int * const p3 = &ci;
```

Oggetti volatili

- La keyword «volatile» specifica che l'oggetto potrebbe cambiare in modo incontrollato
- Le operazioni di lettura e scrittura non vengono ottimizzate

Altre costanti

- Macros vs. `constexpr`
 - Scope
 - Type safety
 - Pre-processore



```
#define ZOP 23  
  
constexpr unsigned int zap = 23;
```

Const vs. Constexpr

- La keyword «`const`»
 - È un *type qualifier*
 - L'oggetto è «non mutabile»: quando assume un valore, questo non può più cambiare
 - Ma i membri «`mutable`» di un oggetto costante possono cambiare
- La keyword «`constexpr`»
 - È uno *specifier*
 - Il valore della espressione può essere determinato durante la compilazione

Suggerimenti



- *Non utilizzate **#define**, ma **constexpr** per le costanti*
- *Utilizzate tutti i **const** che potete*

Programmi su più file

- Scope / Visibilità
 - Dove è possibile utilizzare un nome (nel codice sorgente)
 - Keyword «extern»
 - Keyword «static»
- File include

extern_1.cpp



```
#include <iostream>

void var(int v);
int var();

int main() {
    var(42);
    std::cout << var() << std::endl;
    std::cout << var_ << std::endl;    // Errore!
    return 0;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

79

extern_2.cpp



```
static int var_;

void var(int v) {
    var_ = v;
}

int var() {
    return var_;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

80

lvalue & rvalue

```
int &var_lref() {
    return var_;
}
```

```
int main() {
    var(42);
    std::cout << var() << std::endl;
    var_lref() = 17;
    std::cout << var() << std::endl;
    return 0;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

81

81

lvalue & rvalue



```
int& foo() {
    static int var;
    return var;
}

int foo2() {
    static int var;
    return var;
}

int main() {
    foo() = 42;           // foo() usata come lvalue
    int bar = foo();      // foo() usata come rvalue
    //foo2() = 42;        // error: foo2() non è un lvalue
    int bar2 = foo2();    // foo2() usata come rvalue
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

82

82

Ivalue & rvalue

- Un riferimento a lvalue si indica con «&» nella dichiarazione

```
int& lval = ★;
```

- Un riferimento a rvalue si indica con «&&» nella dichiarazione

```
int&& rval = ★;
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

83

83

lvalue & rvalue reference



```
void f(int& i) {
    std::cout << "f(int& i) : i = " << i << std::endl;
}
void f(int&& i) {
    std::cout << "f(int&& i) : i = " << i << std::endl;
}

int main() {
    int a = 0;
    f(a);
    f(42);
}
```

```
f(int& i) : i = 0
f(int&& i) : i = 42
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

84

84

lvalue & rvalue reference



```
void f(int& i) {
    std::cout << "f(int& i) : i = " << i << std::endl;
}
void f(int&& i) {
    std::cout << "f(int&& i) : i = " << i << std::endl;
}
void f(int i) {
    std::cout << "f(int i) : i = " << i << std::endl;
}

int main() {
    int a = 0;
    f(a);
    f(42);
}
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

85

Array

```
double a[4] = { 1, 2, 3, 4};
for(size_t i = 0; i < 4; ++i)
    std::cout << a[i] << " ";
std::cout << std::endl;
```

- «Vecchio stile» (come nel C degli anni 70)
- Dimensione fissa
- Nessun controllo

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

86

Array

```
double a2[] = { 1, 2, 3, 4};  
for(auto& e : a2)  
    std::cout << e << " ";  
std::cout << std::endl;
```

- «Nuovo stile» (C++11)
- Dimensione fissa
- Nessun controllo

Vettori

```
std::vector<double> v = {2, 3, 4, 5};
```

- «Vettore di double», il tipo degli elementi del vettore è fra parentesi angolari

Vettori e cicli for (classici)

```
double avg2(std::vector<double> v) {
    double sum{0};
    for(int i = 0; i < v.size(); ++i)
        sum += v[i];
    return sum / v.size();
}
```

- int? o auto? o size_t?

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

89

89

Vettori e cicli for (C++)

```
double avg2(std::vector<double> v) {
    double sum{0};
    for(auto p = v.begin(); p != v.end(); ++p)
        sum += *p;
    return sum / v.size();
}
```

- auto p? cosa è p?
- cosa è l'asterisco *p?

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

90

90

Vettori e cicli for moderni (range)

```
double avg2(std::vector<double> v) {
    double sum{0};
    for(auto e: v)
        sum += e;
    return sum / v.size();
}
```

- auto e?

Vettori

- Dichiarazione
- Accedere agli elementi
- begin()
- end()
- size()
- push_back()

std::array

```
std::array<double, 4> m { 2.3, 3.4, 4.5 };
for(auto &e : m)
    std::cout << e << " ";
std::cout << std::endl;
```

- Una rimpiazzo per l'array anni '70
 - Dimensione fissa
 - «ugualmente» veloce e leggero
 - Sintassi un po' più complicata di std::vector

Suggerimenti

- Usate `std::vector` per tutti i problemi complicati (e.g., numero di elementi variabile)
- Usate `std::array` nei casi non-banali (se serve particolare efficienza)
- Usate i «vecchi» array il meno possibile, e in casi molto speciali



Scope

- Namespaces
- «using directive» vs. «using declaration»
 - using namespace foo (directive)
 - using foo::bar (declaration)



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

95

95

Using

```
void f();
namespace A {
    void g();
}
namespace X {
    using ::f;      // f (globale) è visibile come ::X::f
    using A::g;     // A::g è visibile come ::X::g
}
int main() {
    using namespace A;
    g();           // chiama A::g
    X::f();        // chiama ::f
    X::g();        // chiama A::g
    return 0;
}
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

96

96

Pronuncia



- *std*
 - *Standard*
 - *Es-tii-dii*
 - *St'ed*

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

97

97

Duplicati

- Generare 10,000 numeri «causalì» fra 0 e 999

```
int numero = rand()
```

- Contare i duplicati

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

98

98

Duplicati

- Generare 10,000 numeri «causalì» fra 0 e 999

```
int numero = rand()
```

- Contare i duplicati
- Rimuovere i duplicati

Overload!

```
double avg(std::vector<double> v) {
    double sum{0};
    for(auto e: v)
        sum += e;
    return sum / v.size();
}
```

- Perché usare un altro nome? I parametri distinguono la funzione...

Vettori

- Leggere numeri interi da tastiera fino all'inserimento di -1 e memorizzarli in un vettore, ignorare i duplicati. Al termine stampare il contenuto del vettore e la media (utilizzare una funzione).

Puntatori a funzione

- Una variabile può contenere una funzione (un *puntatore a funzione*)

```
void sum(int a, int b) {
    std::cout << a << " + " << b << " = "
        << a + b << std::endl;
}

int main() {
    void (*f)(int, int) = sum;
    f(3, 2);
}
```

Puntatori a funzione

- Se il tipo di un puntatore a funzione sembra troppo complesso, si può sempre usare *auto*

```
void sum(int a, int b) {
    std::cout << a << " + " << b << " = "
                  << a + b << std::endl;
}

int main() {
    auto f = sum;
    f(3, 2);
}
```

Lambda function

- Una funzione lambda è una funzione senza nome

```
auto f = [](int a, int b) { return a+b; };
```

Lambda function e cattura

- Cattura una o più (o tutte) le variabili del chiamante in sola lettura *by value* (=) o *by reference* (&)

```
int a{42};
auto f2 = [&](int x1, int x2) {
    a = x1+x2; return a;
};
std::cout << a << std::endl;
std::cout << f2(3, 2) << std::endl;
std::cout << a << std::endl;
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

105

105

Classi e Oggetti

Una veloce introduzione al C++

Giovanni Squillero
squillero@polito.it

106

Programmazione Orientata agli Oggetti

- Strutture e funzioni
- Classi vs. Oggetti
- Astrazione vs. «Notazione ungherese»
- Incapsulamento



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

107

107

Classi e Oggetti



```
class Agente {  
public:  
    std::string nome;  
    std::string cognome;  
    int anni = 0;  
    double peso = 0;  
};  
  
int main() {  
    Agente james { "James", "Bondi", 42, 79.8 };  
    std::cout << "Mi chiamo " << james.cognome << ", " << james.nome << " "  
        << james.cognome << "." << std::endl;  
    return 0;  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

108

108

Classi e Oggetti



```
class Agente {  
public:  
    std::string nome;  
    std::string cognome;  
    int anni = 0;  
    double peso = 0;  
  
    void saluto() {  
        std::cout << "Mi chiamo " << cognome << ", " << nome << " " << cognome << "."  
        << std::endl;  
    }  
};  
  
int main() {  
    Agente james { "James", "Bond", 42, 79.8 };  
    james.saluto();  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

109

Classi e Oggetti



```
class Agente {  
public:  
    std::string nome;  
    std::string cognome;  
    int anni = 0;  
    double peso = 0;  
  
    void saluto() {  
        std::cout << "Mi chiamo " << cognome << ", " << nome << " " << cognome << "."  
        << std::endl;  
    }  
};  
  
int main() {  
    Agente james { "James", "Bond", 42, 79.8 };  
    james.saluto();  
}
```

A loro volta oggetti...

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

110

Data hiding / encapsulation

- Membri pubblici (public)
 - Tutti possono accedere
- Membri privati (private)
 - Accessibili solo dall'interno della classe stessa (default)
- Membri protetti (protected)
 - ... *prossimamente*
- (Retroattivamente)
 - Una «struct» è una classe con tutti i membri pubblici

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

111

111

const

- Una funzione membro definita `const` non può modificare l'oggetto
- I membri definiti `mutable` possono essere modificati anche da funzioni `const`

```
class Test {
    mutable int fluffy;
    int num;
public:
    void reset() const {
        fluffy = 0;
    }
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

112

112

const

- ... e in ogni caso, posso sempre rimuovere const con un const_cast

```
class Test {  
    mutable int fluffy;  
    int num;  
public:  
    void reset() const {  
        fluffy = 0;  
        const_cast<Test*>(this)->num = 0;  
    }  
};
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

113

Promemoria

- ::
 - Namespace
 - Classi
- .
 - Oggetti
- ->
 - Puntatori a oggetti
 - $x->y \approx (*x).y$

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

114

Data hiding



```
class Agente {  
    int anni = 0;  
    double peso = 0;  
public:  
    std::string nome;  
    std::string cognome;  
};  
  
int main() {  
    Agente james { "James", "Bond" };  
    std::cout << "Mi chiamo " << james.cognome << ", " << james.nome << " "  
        << james.cognome << "." << std::endl;  
    return 0;  
}
```

error: no matching function for call to
Agente::Agente(<brace-enclosed initializer list>)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

115

Classi e funzioni Amiche

- Una classe può dichiarare funzioni amiche (friend)
 - Queste funzioni possono accedere ai membri della classe come se fossero a loro volta dei membri
- Una classe può dichiarare classi amiche (friend)
 - Tutti i metodi di queste classi diventano funzioni amiche

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

116

Puntatori a Oggetti

- Come per ogni altra variabile, esistono *puntatori* a oggetti

```
class Cane;  
  
Cane fido;  
Cane* puntatore_a_fido = &fido;
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

117

117

this

- Nele funzioni membro this è il puntatore all'oggetto corrente

```
class Cane {  
private:  
    std::string nome;  
public:  
    void set_nome(std::string nome) {  
        this->nome = nome;  
    }  
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

118

118

Setter & Getter

```
class Cane {  
private:  
    std::string nome;  
public:  
    void set_nome(std::string nome) {  
        this->nome = nome;  
    }  
};
```



```
class Cane {  
private:  
    std::string nome_;  
public:  
    void nome(std::string n) {  
        nome_ = n;  
    }  
};
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

119

Classe «Data»

- Trasformare la struttura «Data» in una classe.
- Aggiungere il metodo `stampa()` per stampare la data nel formato: «giorno-mese-anno» (e.g., 17-3-1980)

```
struct Data {  
    int anno;  
    int mese;  
    int giorno;  
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

120

Classe «Data»

```
class Data {  
    int anno_;  
    int mese_;  
    int giorno_;  
public:  
    int anno() { return anno_; }  
    void anno(int a) { anno_ = a; };  
    int mese() { return mese_; }  
    void mese(int m) { mese_ = m; };  
    int giorno() { return giorno_; }  
    void giorno(int g) { giorno_ = g; };  
    void stampa() {  
        std::cout << giorno_ << "-" << mese_ << "-" << anno_ << std::endl;  
    }  
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp



121

Costruttore

```
class Data {  
    int anno_;  
    int mese_;  
    int giorno_;  
public:  
    Data(int a, int m, int g) : anno_{a}, mese_{m}, giorno_{g} {};  
  
int main() {  
    Data natale {2018, 12, 25};  
    natale.stampa();  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp



122

122

Costruttore

```
class Data {  
    int anno_;  
    int mese_;  
    int giorno_;  
public:  
    Data(int a, int m, int g) : anno_{a}, mese_{m}, giorno_{g} {};  
};  
  
int main() {  
    Data natale {2018, 12, 25};  
    natale.stampa();  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

123

123

Costruttori

- Default constructor
- Explicit constructor
- Copy constructor (Converting constructors)
- Move constructor

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

124

124

Copy constructor

- Unico parametro: `T&, const T&, volatile T&`,
o `const volatile T&`
- Inizializzazione
 - `T foo = bar`
 - `T foo(bar)`
- Chiamata di funzione (argomento passato per valore)
 - `f(foo)`
- Valore di ritorno
 - `return foo`

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

125

125

Costruttori

- Il costruttore viene utilizzato durante cast e «copy initialization»

```
struct Foo {
    int val;
    Foo(int v) : val{v} { }
    Foo(int v1, int v2) : val{v1 + v2} { }      // c++11
};

int main() {
    Foo a = 1;          // copy initialization
    Foo b(2);
    Foo c{3};
    Foo d{4, 5};
    Foo e{5, 6};
    Foo f = {5, 6};    // copy initialization
    Foo g = (Foo)7;
    ...
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

126

126

Non «Converting» Constructors

- La keyword `explicit` impedisce che il costruttore venga utilizzato implicitamente per una «copy initialization»

```
struct Foo {
    int val;
    explicit Foo(int v) : val{v} { }
    explicit Foo(int v1, int v2) : val{v1 + v2} { } // c++11
};

int main() {
    Foo a = 1;           // copy initialization
    Foo b(2);
    Foo c{3};
    Foo d(4, 5);
    Foo e{5, 6};
    Foo f = {5, 6};     // copy initialization
    Foo g = (Foo)7;
    ...
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

127

Move constructor

- Unico parametro: `T&&`, `const T&&`, `volatile T&&`,
o `const volatile T&&`
- Inizializzazione
 - `T foo = std::move(bar)`
 - `T foo(std::move(bar))`
- Chiamata di funzione (argomento passato per valore)
 - `f(std::move(foo))`
- Valore di ritorno
 - `return foo`

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

128

Distruttore



```
class Data {  
    int anno_;  
    int mese_;  
    int giorno_;  
public:  
    Data(int a, int m, int g) : anno_{a}, mese_{m}, giorno_{g} { };  
    ~Data() { std::cerr << "Goodbye..." << std::endl; };  
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

129

129

Aggregate Initialization

- Una forma di «list initialization» per dati aggregati

```
class Punto {  
public:  
    int x, y;  
};  
  
int main() {  
    Punto origine = {0, 0};  
    Punto zap{23, 10};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

130

130

Stampa()

- Fare in modo che natale.stampa() stampi sullo schermo
25-dicembre-2018
- Suggerimento: usare una struct dichiarata dentro la classe per convertire il numero del mese in stringa

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

131

131

Errori

- Tipi di errori
 - Compile-time:
 - di sintassi
 - di tipo
 - Link-time
 - Run-time
 - rilevati dal computer (hardware o sistema operativo)
 - rilevati da una libreria
 - rilevati dall'utente
 - Logici

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

132

132

Classe «Data»

- Controllare se la data è corretta nel costruttore
 - Notez bien: è necessario memorizzare la lunghezza di ogni mese
 - Suggerimento: nella struttura che contiene il nome dei mesi, memorizzare anche la lunghezza

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

133

133

Data

```
class Data {  
  
protected:  
    struct Mese {  
        std::string nome;  
        int lunghezza;  
    };  
  
    std::array<Mese, 13> calendario_ {  
        "?", 0,  
        "gennaio", 31, "febbraio", 28, "marzo", 31, "aprile", 30, "maggio", 31,  
        "giugno", 30, "luglio", 31, "agosto", 31, "settembre", 30, "ottobre", 31,  
        "novembre", 30, "dicembre", 31,  
    };
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

134

134

Data



```
bool ok() {
    if(mese_ < 1 || mese_ > 12)
        return false;
    if(giorno_ < 1 || giorno_ > calendario_[mese_].lunghezza)
        return false;
    if(anno_ < 1)
        return false;
    return true;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

135

135

Data



```
private:
    int anno_;
    int mese_;
    int giorno_;

public:
    Data(int a, int m, int g) : anno_{a}, mese_{m}, giorno_{g} {
        if(!ok()) {
            std::cerr << "Data illegale: "
                  << giorno_ << "-" << mese_ << "-" << anno_ << std::endl;
        }
    };
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

136

136

Data



```
    friend std::ostream &operator<< (std::ostream &os, Data const &d);  
};  
  
std::ostream &operator<< (std::ostream &os, Data const &d) {  
    os << d.giorno_ << "-" << d.calendario_[d.mese_].nome << "-" << d.anno_;  
    return os;  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

137

Throw

- Il programma può segnalare un errore mediante una *eccezione*
- Meccanismo comune a tutti i linguaggi moderni (Java, Python, Go, ...)
- Il chiamante può decidere come comportarsi
- Non è necessario controllare espressamente le condizioni di errore ogni volta

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

138

Throw

- Si utilizza una classe per gestire/descrivere l'errore

```
class Bad_date { };

class Data {
[...]
    Data(int a, int m, int g)
        : anno_{a}, mese_{m}, giorno_{g} {
            if(!ok())
                throw Bad_date{ };
    };
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

139

139

Try / Catch

- Aggiungo un costruttore di default

```
Data() : anno_{0}, mese_{0}, giorno_{0} { };
```

- Modifico il main

```
Data bad_date;
try {
    bad_date = Data{ 2018, 2, 29 };
} catch(Bad_date& d) {
    std::cout << "Pazienza..." << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

140

140

Exceptions

- Le eccezioni possono rendere non *totalmente* predicibile il comportamento del programma
- In un sistema real-time (critico) non dovrebbero essere usate.

Classe «Data»

- Aggiungere il controllo per gli anni bisestili
- Nel calendario gregoriano sono bisestili:
 - gli anni non secolari il cui numero è divisibile per 4
 - gli anni secolari il cui numero è divisibile per 400
- Esempi:
 - Il 1896 è bisestile (divisibile per 4 ma non per 100)
 - Il 1900 non è bisestile (divisibile per 4 e per 100, ma non per 400)
 - Il 2000 è bisestile (divisibile per 4, 100 e 400)

Enum «vecchio stile»



```
enum { gennaio = 1, febbraio, marzo, aprile, maggio, giugno, luglio,
       agosto, settembre, ottobre, novembre, dicembre };

int main() {
    int i = gennaio;
    i += 2;
    i -= marzo;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

143

Enum «vecchio stile»



```
enum Mese { gennaio = 1, febbraio, marzo, aprile, maggio, giugno, luglio,
            agosto, settembre, ottobre, novembre, dicembre };

int main() {
    Mese m = 3;           // Errore
    int m = gennaio;     // Ok
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

144

Class Enum del C++11



```
enum class Mese { gennaio = 1, febbraio, marzo, aprile, maggio, giugno, luglio,
                  agosto, settembre, ottobre, novembre, dicembre };

int main() {
    Mese m1 = Mese::gennaio;
    Mese m2{Mese::febbraio};
    Mese m3 = Mese(1);           // ok, ma non controllato

    int mi = Mese::gennaio;     // type safety: errore di conversione Mese -> int
    Mese m4{2};                 // type safety: errore di conversione int -> Mese
    Mese m5(2);                 // errore: costruttore Mese(int) non definito
    Mese m6 = Mese{9};          // type safety: errore di conversione int -> Mese
    m1 += 1;                    // type safety: errore di conversione int -> Mese
    m1 += Mese::febbraio;       // errore: operatore += non definito per Mese
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

145

145

Conversioni

- Da «Mese» a «int» (sicura)

```
int mese2int(Mese m) {
    return static_cast<int>(m);
}
```

- Da «int» a «Mese» (controllata)

```
Mese int2mese(int x) {
    if (x < static_cast<int>(Mese::gennaio)
        || x > static_cast<int>(Mese::dicembre))
        throw Eccezione {};
    return N(x);
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

146

146

Overloading degli Operatori

Espressione	Metodo	Funzione
$\star a$	(a).operator★()	operator★(a)
$a \star b$	(a).operator★(b)	operator★(a, b)
$a = b$	(a).operator=(b)	
$a(b\dots)$	(a).operator()(b\dots)	
$a[b]$	(a).operator[](b)	
$a->$	(a).operator->()	
$a \star$	(a).operator★(0)	operator★(a, 0)

Operatori Comuni

Assegnazione	Incremento Decremento	Aritmetici	Logici	Confronto	Accesso ai membri	Altri
$a = b$		+a -a				
$a += b$		a + b				
$a -= b$		a - b				
$a *= b$		a * b				
$a /= b$	++a	a / b	!a	a == b a != b	a[b] *a	
$a -= b$	--a	a % b	a && b	a < b a > b	&a a->b	
$a \% b$	a++	a % b	a b	a <= b a >= b	a.b a->*b	
$a \&= b$	a--	~a		a <= b a >= b	? :	
$a = b$		a & b		a >= b		
$a ^= b$		a b		a <= b		
$a <= b$		a ^ b				
$a >= b$		a << b				
$a >> b$		a >> b				

Overloading di +=

- Fare in modo che sia possibile «incrementare» il mese di una quantità intera

```
Mese m{Mese::N::ottobre};  
std::cout << int(m.mese()) << std::endl;           // stampa 10  
m += 1;  
std::cout << int(m.mese()) << std::endl;           // stampa 11  
m += 99;                                            // genera una eccezione!
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

149

149

cppreference.com

Page Discussion View Edit History Actions

C++ C++ language Expressions

Assignment operators

Assignment operators modify the value of the object.

Operator name	Syntax	Overloadable	Prototype examples (for <code>class T</code>)	
			Inside class definition	Outside class definition
simple assignment	<code>a = b</code>	Yes	<code>T& T::operator =(const T2& b);</code>	N/A
addition assignment	<code>a += b</code>	Yes	<code>T& T::operator +=(const T2& b);</code>	<code>T::operator +=(T& a, const T2& b);</code>
subtraction assignment	<code>a -= b</code>	Yes	<code>T& T::operator -=(const T2& b);</code>	<code>T::operator -(=)(T& a, const T2& b);</code>
multiplication assignment	<code>a *= b</code>	Yes	<code>T& T::operator *(=)(const T2& b);</code>	<code>T::operator *=(T& a, const T2& b);</code>
division assignment	<code>a /= b</code>	Yes	<code>T& T::operator /=(const T2& b);</code>	<code>T::operator /=(T& a, const T2& b);</code>
modulus assignment	<code>a %= b</code>	Yes	<code>T& T::operator %=(const T2& b);</code>	<code>T::operator %=(T& a, const T2& b);</code>
bitwise AND assignment	<code>a &= b</code>	Yes	<code>T& T::operator &=(const T2& b);</code>	<code>T::operator &=(T& a, const T2& b);</code>
bitwise OR assignment	<code>a = b</code>	Yes	<code>T& T::operator =(const T2& b);</code>	<code>T::operator =(T& a, const T2& b);</code>
bitwise XOR assignment	<code>a ^= b</code>	Yes	<code>T& T::operator ^=(const T2& b);</code>	<code>T::operator ^=(T& a, const T2& b);</code>
left shift assignment	<code>a <<= b</code>	Yes	<code>T& T::operator <<=(const T2& b);</code>	<code>T::operator <<=(T& a, const T2& b);</code>
right shift assignment	<code>a >>= b</code>	Yes	<code>T& T::operator >>=(const T2& b);</code>	<code>T::operator >>=(T& a, const T2& b);</code>

Notes

- All built-in assignment operators return `*this`, and most user-defined overloads also return `*this` so that the user-defined operators can be used in the same manner as the built-ins. However, in a user-defined operator overload, any type can be used as return type (including `void`).
- `T2` can be any type including `T`

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

150

150

Overloading di +=



```
class Mese {  
public:  
    enum class N { sconosciuto = 0, gennaio = 1, febbraio, marzo, ..., dicembre };  
protected:  
    N int2mese(int x) {  
        if (x < static_cast<int>(N::sconosciuto) || x > static_cast<int>(N::dicembre))  
            throw No_data { };  
        return N(x);  
    }  
public:  
    Mese& operator+=(const int& inc) {  
        mese_ = int2mese(static_cast<int>(mese_) + inc);  
        return *this;  
    }  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

151

Overloading di <<

- Fare in modo che «cout << natale» stampi sullo schermo

25-dicembre-2018

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

152

152

Overloading di <<

- Suggerimento:

– << e >> sono gli operatori di shift ridefiniti per la classe ostream

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

153

153

- Suggerimento:

— <<

Arithmetic operators
Returns the result of specific arithmetic operation.

Operator name	Syntax	Overloadable	Prototype examples (for class T)
unary plus	+a	Yes	T T::operator+() const; T T::operator+(const T &a);
unary minus	-a	Yes	T T::operator-() const; T T::operator-(const T &a);
addition	a + b	Yes	T T::operator+(const T2 &b) const; T T::operator+(const T &a, const T2 &b);
subtraction	a - b	Yes	T T::operator-(const T2 &b) const; T T::operator-(const T &a, const T2 &b);
multiplication	a * b	Yes	T T::operator*(const T2 &b) const; T T::operator*(const T &a, const T2 &b);
division	a / b	Yes	T T::operator/(const T2 &b) const; T T::operator/(const T &a, const T2 &b);
modulo	a % b	Yes	T T::operator%(const T2 &b) const; T T::operator%(const T &a, const T2 &b);
bitwise NOT	~a	Yes	T T::operator~() const; T operator~(const T &a);
bitwise AND	a & b	Yes	T T::operator&(const T2 &b) const; T operator&(const T &a, const T2 &b);
bitwise OR	a b	Yes	T T::operator (const T2 &b) const; T operator (const T &a, const T2 &b);
bitwise XOR	a ^ b	Yes	T T::operator^(const T2 &b) const; T operator^(const T &a, const T2 &b);
bitwise left shift	a << b	Yes	T T::operator<<(const T2 &b) const; T operator<<(const T &a, const T2 &b);
bitwise right shift	a >> b	Yes	T T::operator>>(const T2 &b) const; T operator>>(const T &a, const T2 &b);

Notes

- All built-in operators return values, and most user-defined overloads also return values so that the user-defined operators can be used in the same manner as the built-ins. However, in a user-defined operator overload, any type can be used as return type (including `void`). In particular, stream insertion and stream extraction overloads of operator<< and operator>> return `T`.
- `T2` can be any type including `T`

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

154

154

Overloading di <<

- Per rendere possibile «cout << data» bisogna definire

```
std::ostream& operator<< (std::ostream& os, Data const & d)
```



- ... e ricordate le funzioni «friend»

Funzioni automatiche

- Alcune funzioni membro sono generate automaticamente, ma in C++11 è possibile modificare:
 - Il tipo di accesso (*public, private, protected*)
 - Renderle virtuali
 - Renderle esplicite (ripristinare un costruttore)
 - Aggiungere/rimuovere *const* dai parametri

= default



```
class Agent {  
private:  
    std::string id_;  
public:  
    Agent() = default;  
    Agent(std::string id) : id_{id} {}  
    virtual ~Agent() = default;  
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

157

157

Rimuovere funzioni

- In C++11 è possibile disabilitare la generazione automatica di alcune funzioni membro

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

158

158

= delete



```
class Agent {  
    Agent(const Agent&) = delete;  
    Agent& operator=(const Agent&) = delete;  
};  
  
int main() {  
    Agent jack_bond = Agent{"007"}; // Errore: usa Agent(const Agent&)  
  
    Agent bob_rock;  
    bob_rock = james_bond; // Errore: usa Agent& operator=(const Agent&)  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

159

159

Funzioni automatiche

- Attenzione, le funzioni automatiche possono venir cancellate per motivi non banali



Deleted implicitly-declared copy constructor

The implicitly-declared copy constructor for class T is undefined if any of the following conditions are true:

(until C++11)

The implicitly-declared or defaulted copy constructor for class T is defined as *deleted* if any of the following conditions are true:

(since C++11)

- T has non-static data members that cannot be copied (have deleted, inaccessible, or ambiguous copy constructors);
- T has direct or virtual base class that cannot be copied (has deleted, inaccessible, or ambiguous copy constructors);
- T has direct or virtual base class with a deleted or inaccessible destructor:
 - T has a user-defined move constructor or move assignment operator;
 - T is a union-like class and has a variant member with non-trivial copy constructor; (since C++11)
 - T has a data member of rvalue reference type

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

160

160

Refactoring

- Riscrivere la classe Data
 - 3 classi membro (Giorno, Mese, Anno)
 - 2 file per classe intestazione + implementazione
 - Mettere controlli di validità
 - Nella classe Mese utilizzare «class enum»



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

161

161

Override operatori

- Tutti gli operatori possono essere ridefiniti per utilizzare le classi
- Leggibilità?

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

162

162

Classe «Ora»

- Scrivere la classe «Ora»
 - Costruttore: Ora() e Ora(int ore, int min, int sec)
 - Eccezione se non corretta

Classe «Ora»

- Metodi «h12» e «h24»

```
char buf[8 + 1];
sprintf(buf, "%02d:%02d:%02d", 1, 2, 3);
return std::string{buf};
```
- Nota
 - Dopo le 11:59:59AM viene 12:00:00PM (mezzogiorno)
 - Dopo le 11:59:59PM viene 12:00:00AM (mezzanotte)
 - Dopo le 23:59:59 viene 24:00:00 (mezzanotte)
 - Dopo 24:00:00 viene 00:00:01 (00:00:00 non esiste)

Ereditarietà



```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class B : public A
{
    // x è public
    // y è protected
    // z non è accessibile da B
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

165

Ereditarietà



```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class C : protected A
{
    // x è protected
    // y è protected
    // z non è accessibile da C
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

166

Ereditarietà



```
class A
{
public:
    int x;
protected:
    int y;
private:
    int z;
};

class D : private A    // default
{
    // x è private
    // y è private
    // z non è accessibile da D
};
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

167

Anniversario

- Costruire la classe Anniversario che
 - «estende» Data
 - «estende» Ora
 - aggiunge un campo che descrive l'anniversario

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

168

Anniversario



```
class Anniversario : protected Data {
    std::string descrizione;
public:
    Anniversario(int a, int m, int g, std::string d) : Data{a, m, g}, descrizione{d} { }
    void stampa() {
        Data::stampa();
        std::cout << descrizione << std::endl;
    }
};

int main() {
    Anniversario natale {2018, 12, 25, "Natale!" };
    natale.stampa();
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

169

169

Polimorfismo

```
struct Frutta {
    string qualita;
    Frutta(string q) : qualita{q} { }
    virtual void tipo() {
        cout << "Frutta " << qualita << endl;
    }
};

struct Pera : Frutta {
    Pera(string q) : Frutta{q} { }
    void tipo() {
        cout << "Pera " << qualita << endl;
    }
};

struct Mela : Frutta {
    Mela(string q) : Frutta{q} { }
    void tipo() {
        cout << "Mela " << qualita << endl;
    }
};
```

squillero@polito.it

170

170

Polimorfismo

- Trovare l'errore

```
int main() {
    Frutta f1 = Mela{"golden"};
    Frutta f2 = Pera{"ruset"};
    f1.tipo();
    f2.tipo();
    ...
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

171

171

Polimorfismo

- Le funzioni devono essere dichiarate virtuali
- Bisogna usare puntatori o riferimenti (vtable)

```
struct Frutta {
    string qualita;
    Frutta(string q) : qualita{q} { }
    virtual void tipo();
};

int main() {
    std::array<Frutta*, 2> f;
    f[0] = new Mela{"golden"};
    f[1] = new Pera{"ruset"};
    for(auto a : f)
        a->tipo();
    ...
}
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

172

172

Classe «Anniversario»

- Modificare «Data» e «Ora» rendendo il metodo «stampa» virtuale
- Aggiungere «stampa» ad «Anniversario»
- Costruire un vettore di puntatori a Ora/Data/Anniversario e stampare il contenuto

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

173

173

Interfacce

- Una «interfaccia» è una classe con almeno una funzione «virtuale pura»

```
struct Frutta {  
    string qualita;  
    Frutta(string q) : qualita{q} { }  
    virtual ~Frutta() = default;  
    virtual void tipo() = 0;  
};
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

174

174

Conversione

- Le conversioni fra tipi sono controllabili

```
struct Zap {
    int val;
    operator std::string () const { return "Sono una stringa!"; }
};

int main(void) {
    Zap x{42};
    std::cout << static_cast<std::string>(x) << std::endl;
...
}
```

Puntatori

Una veloce introduzione al C++

Tipi di Memoria

- Statica
 - Heap (variabili *globali* e *statiche*)
 - Stack (variabili *locali*)
- Dinamica
 - Heap (malloc/free)
 - Free-store (new/delete)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

177

177



Heap/Stack

- Scope: block vs. file
- Lifespan: auto vs. static

```
#include <cstdlib>

int Global_variable = 42;           ← Yellow arrow from 'Global_variable'

void foo(int bar) {                ← Orange arrow from 'bar'
    int local_variable;           ← Yellow arrow from 'local_variable'
    static int local_but_static;  ← Yellow arrow from 'local_but_static';

}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

178

178



heap

Malloc/Free

- Con `malloc` è possibile chiedere al sistema operativo un blocco di memoria contigua di una certa dimensione (in byte)

```
#include <cstdlib>

int main() {
    int* p = (int *)std::malloc(sizeof (int) * 1024);
    for(size_t t = 0; t < 1024; ++t)
        p[t] = t;
}
```

- La memoria non è inizializzata (ma esiste `calloc`)
- Problemi di type checking, cast, ...
- Bisogna ricordarsi di liberare la memoria con `free`

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

179



free-store

New/Delete

- Con `new` è possibile chiedere al sistema operativo un nuovo oggetto

```
int main() {
    Cane* c = new Cane("Fido");
}
```

- La memoria è inizializzata
- Type safety
- Bisogna ricordarsi di liberare la memoria con `delete`

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

180

Factory: new/delete



```
class Cane {
public:
    std::string nome;
    ~Cane() { std::cout << "Addio " << nome << std::endl; }
};

Cane* factory(std::string name) {
    Cane* c = new Cane{name};
    return c;
}

int main() {
    auto c = factory("Fido");
    std::cout << "Ciao " << c->nome << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

181

181

Problema

- **Problema:**
 - Al termine del programma la memoria non viene rilasciata
- **Soluzione:** `unique_ptr<>` e `shared_ptr<>`
 - Un oggetto che gestisce un altro oggetto attraverso un puntatore, e smaltisce l'oggetto puntato
 - Unisce flessibilità di `new/delete` alla praticità delle variabili automatiche

```
std::unique_ptr<C> v{new C{...}};  
auto v = std::make_unique<C>(...);
```

C++
14

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

182

182



unique_ptr

```
class Cane {
public:
    std::string nome;
    ~Cane() { std::cout << "Addio " << nome << std::endl; }
};

std::unique_ptr<Cane> factory2(std::string name) {
    std::unique_ptr<Cane> c{new Cane{name}};
    return c;
}

int main() {
    auto c = factory2("Fido");
    std::cout << "Ciao " << c->nome << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

183

183

Unique pointers

- Gli *unique pointer* definiscono chi *possiede* l'oggetto (*ownership*)
- L'oggetto viene distrutto con l'*up*, o quando viene perso il riferimento (e.g., *up* = *nullptr*)
- Non è possibile assegnare un *up* ad un altro
- La funzione *get* ritorna il puntatore grezzo (*raw pointer*)
- La funzione *release* ritorna il puntatore grezzo, e libera l'oggetto — lo *up* non è più utilizzabile

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

184

184

Unique pointers

- Gli *unique pointer* definiscono chi possiede l'oggetto (A *unique_ptr* has the interesting property of having *no overhead compared to an ordinary pointer*)
- L'interesting property of having *no overhead compared to an ordinary pointer*, o quando viene perso il riferimento — lo *up* non è più utilizzabile
- La funzione *get* ritorna il puntatore grezzo (*raw pointer*)
- La funzione *release* ritorna il puntatore grezzo, e libera il riferimento — lo *up* non è più utilizzabile



https://github.com/squillero/introduzione_al_cpp

185

185

Unique pointers



```
void gioco(std::unique_ptr<Cane> c) {
    std::cout << "Gioco con " << c->nome << std::endl;
}

void gioco(Cane* c) {
    std::cout << "Gioco con " << c->nome << std::endl;
}

int main() {
    auto c = factory2("Fido");
    std::cout << "Ciao " << c->nome << std::endl;
    gioco(c);           // Errore: unique_ptr(const unique_ptr&) = delete
    gioco(c.get());    // ok
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

186

186

Shared pointers

- Come gli unique pointers, ma lo stesso oggetto può essere posseduto da più sp
- L'oggetto viene distrutto con l'ultimo *sp*, o quando viene perso l'ultimo riferimento
- Maggior overhead

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

187

187

Shared pointers

```
std::shared_ptr<Cane> factory3(std::string name) {
    std::shared_ptr<Cane> c{new Cane{name}};
    return c;
}

int main() {
    auto c = factory3("Fido");
    std::cout << "Ciao " << c->nome << std::endl;
{
    auto c2 = c;
    gioco(c2.get());
}
gioco(c.get());
std::cout << "That's all folks..." << std::endl;
}
```



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

188

188

Riferimenti ciclici

```
struct E {
    std::shared_ptr<E> link;
};

int main() {
    auto e1 = std::make_shared<E>();
    auto e2 = std::make_shared<E>();
    e1->link = e2;
    e2->link = e1;
}
```

- Soluzione: `weak_ptr`!
 - Riferimento «non proprietario» di uno shared pointer
 - Sicuro

Caveat

- Frammentazione della memoria
- Real-time

Classe Interna

- Semplice classe per monitorare costruttore/distruttore

```
struct Inner {
    Inner() { std::cout << "New Inner @" << this << std::endl; }
    ~Inner() { std::cout << "Delete Inner @" << this << std::endl; }
};
```

- Notez bien: Le «strutture» sono classi con tutti i membri pubblici

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

191

191

Classe Esterna: Tutto ok

```
struct Outer {
    std::string name;
    Inner* in = new Inner;
    Outer(std::string n) : name{n} {
        std::cout << "New Outer \" " << name << "\" @" << this << std::endl;
    }
    ~Outer() {
        std::cout << "Delete Outer \" " << name << "\" @" << this << std::endl;
        delete in;
    }
};

int main() {
    Outer test{"Foo"};
    return 0;
}
```

New Inner @ 0x27a1ee0
New Outer "Foo" @ 0x62fee8
Delete Outer "Foo" @ 0x62fee8
Delete Inner @ 0x27a1ee0

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

192

192

Assegnazione : Tutto ok

```
int main() {
    Outer test = Outer{"Ok"};
    return 0;
}
```

```
New Inner @ 0xc1ee0
New Outer "Ok" @ 0x62fee8
Delete Outer "Ok" @ 0x62fee8
Delete Inner @ 0xc1ee0
```

- Ma cosa è successo esattamente?

Assegnazione : Problema?

```
int main() {
    Outer left{ "?" };
    Outer right{ "Not_Ok" };
    left = right;
    std::cout << left.name << ":" << left.in << std::endl;
    std::cout << right.name << ":" << right.in << std::endl;
    return 0;
}
```

```
New Inner @ 0x3021ee0
New Outer "?" @ 0x62fecc
New Inner @ 0x3021ef0
New Outer "Not_Ok" @ 0x62feb0
Not_Ok: 0x3021ef0
Not_Ok: 0x3021ef0
```

- Il membro name contiene lo stesso valore
- Sia left sia right contengono un puntatore alla stessa struttura Inner 0x3021ef0
- Inner @ 0x3021ee0 non è più utilizzabile

Copy assignment operator

```
Struct Outer {  
...  
    Outer& operator=(Outer o) {  
        std::cout << "Copy! " << this << " <- " << &o << std::endl;  
        delete in;  
        in = new Inner;  
        // copia il contenuto richiesto da o.in  
        name = o.name;  
        return *this;  
    }  
}
```

- È possibile ridefinire l'operatore di assegnazione (copia)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

195

195

Assegnazione: Problema?

```
int main() {  
    Outer foo{"Foo"};  
    std::cout << "-----" << std::endl;  
    Outer bar{"Bar"};  
    std::cout << "-----" << std::endl;  
    foo = bar;  
    std::cout << "-----" << std::endl;  
}
```

```
New Inner @ 0x2f31ee0  
New Outer "Foo" @ 0x62feb0  
-----  
New Inner @ 0x2f31ef0  
New Outer "Bar" @ 0x62fe94  
-----  
Copy! 0x62feb0 <- 0x62ff04  
Delete Inner @ 0x2f31ee0  
New Inner @ 0x2f31ef0  
Delete Outer "Bar" @ 0x62ff04  
Delete Inner @ 0x2f31ef0  
-----  
Delete Outer "Bar" @ 0x62fe94  
Delete Inner @ 0x2f31ef0  
Delete Outer "Bar" @ 0x62feb0  
Delete Inner @ 0x2f31ee0
```

- Esattamente cosa sto copiando in Bar?
- Chi ha creato Bar @ 0x62ff04?
- Tutto funziona correttamente con Inner

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

196

196

Passaggio di Parametri

- Per valore (*by value*)

```
void foo(Outer o)
```

- Il parametro viene *copiato* in una zona di memoria (stack)
- La funzione viene chiamata
- L'oggetto Bar @ 0x62ff04 è la copia di Bar @ 0x62fe94

- Per riferimento (*by reference*)

```
void foo(const Outer& o)
```

- Viene passato solo il *puntatore* all'oggetto
- Usare `const` se possibile

Copy assignment operator

```
Struct Outer {  
...  
    Outer& operator=(const Outer& o) {  
        std::cout << "Copy! " << this << " - " << &o <<  
        std::endl;  
        delete in;  
        in = new Inner;  
        // copia il contenuto richiesto da o.in  
        name = o.name;  
        return *this;  
    }  
}
```



Assegnazione: Quasi tutto ok

```
int main() {
    Outer foo{"Foo"};
    std::cout << "-----" << std::endl;
    Outer bar{"Bar"};
    std::cout << "-----" << std::endl;
    foo = bar;
    std::cout << "-----" << std::endl;
}
```

```
New Inner @ 0x3081ee0
New Outer "Foo" @ 0x62fecc
-----
New Inner @ 0x3081ef0
New Outer "Bar" @ 0x62feb0
-----
Copy! 0x62fecc <- 0x62feb0
Delete Inner @ 0x3081ee0
New Inner @ 0x3081ee0
-----
Delete Outer "Bar" @ 0x62feb0
Delete Inner @ 0x3081ef0
Delete Outer "Bar" @ 0x62fecc
Delete Inner @ 0x3081ee0
```

- Il corpo della funzione non è stato modificato
- Il main non è stato modificato

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

199

199

Assegnazione: Quasi tutto ok

```
int main() {
    Outer foo{"foo"};
    Outer bar = foo;
    std::cout << "-----" << std::endl;
}
```

```
New Inner @ 0x2fe1ee0
New Outer "foo" @ 0x62fee8
-----
Delete Outer "foo" @ 0x62fecc
Delete Inner @ 0x2fe1ee0
Delete Outer "foo" @ 0x62fee8
Delete Inner @ 0x2fe1ee0
```

- Il problema si presenta ancora se inizializzo bar con foo

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

200

200

Copy constructor

```
Struct Outer {  
...  
    Outer(const Outer& o) {  
        name = o.name;  
        // inizializza in  
        std::cout << "New Outer (copy constructor) \"" << name << "\"" @ "  
                         << this << std::endl;  
    }  
}
```

- Chiamato quando si esegue

```
Outer bar = foo;
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

201

201

Assegnazione: Tutto ok

```
int main() {  
    Outer foo{"foo"};  
    Outer bar = foo;  
    std::cout << "-----" << std::endl;  
    bar = foo;  
    std::cout << "-----" << std::endl;  
}  
  
New Inner @ 0x28a1ee0  
New Outer "foo" @ 0x62fee8  
New Inner @ 0x28a1ef0  
New Outer (copy constructor) "foo" @ 0x62fecc  
-----  
Copy! 0x62fecc <- 0x62fee8  
Delete Inner @ 0x28a1ef0  
New Inner @ 0x28a1ef0  
-----  
Delete Outer "foo" @ 0x62fecc  
Delete Inner @ 0x28a1ef0  
Delete Outer "foo" @ 0x62fee8  
Delete Inner @ 0x28a1ee0
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

202

202

Esercizio

- Modificare la classe Outer per utilizzare uno shared_ptr

```
struct Outer {  
    std::string name;  
    std::unique_ptr<Inner> in{new Inner{}};  
...  
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

203

203

Suggerimento

- Consultate la pagine di cppreference.com su unique_ptr



The screenshot shows the 'unique_ptr' page on cppreference.com. The page is organized into sections: 'Member functions', 'Modifiers', 'Observers', and 'Single-object version, unique_ptr<T>'. Each section lists member functions with their descriptions. A blue arrow points from the slide towards the 'unique_ptr' page.

Member functions	
(constructor)	constructs a new unique_ptr (public member function)
(destructor)	destructs the managed object if such is present (public member function)
operator=	assigns the unique_ptr (public member function)
Modifiers	
release	returns a pointer to the managed object and releases the ownership (public member function)
reset	replaces the managed object (public member function)
swap	swaps the managed objects (public member function)
Observers	
get	returns a pointer to the managed object (public member function)
get_deleter	returns the deleter that is used for destruction of the managed object (public member function)
operator bool	checks if there is an associated managed object (public member function)
Single-object version, unique_ptr<T>	

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

204

204

Memory Layout

```
struct Inner {
    Inner() { std::cout << "New Inner @ " << this << std::endl; }
    ~Inner() { std::cout << "Delete Inner @ " << this << std::endl; }
};

struct Outer {
    Inner in;
    char altri_dati[1024];
    Outer() { std::cout << "New Outer @ " << this << std::endl; }
    ~Outer() { std::cout << "Delete Outer @ " << this << std::endl; }
};

int main() {
    Outer out1;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

205

205

Memory Layout

```
struct Inner {
    Inner() { std::cout << "New Inner @ " << this << std::endl; }
    ~Inner() { std::cout << "Delete Inner @ " << this << std::endl; }
};

struct Outer {
    Inner in;
    char altri_dati[1024];
    Outer() { std::cout << "New Outer @ " << this << std::endl; }
    ~Outer() { std::cout << "Delete Outer @ " << this << std::endl; }
};
```

New Inner @ 0x62ff2f
New Outer @ 0x62ff2f
Delete Outer @ 0x62ff2f
Delete Inner @ 0x62ff2f



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

206

206

reinterpret_cast

```
int main() {
    Outer out1;

    Inner* orrore = reinterpret_cast<Inner*>(&out1);
    delete orrore;
    ...
}
```

- Utilizzare *out1* dopo che l'oggetto membro *in* è stato distrutto genera un comportamento impredicibile
- Ma il C++ lo permette!

static_cast

```
int main() {
    Outer out1;

    void* orrore1 = static_cast<void*>(&out1);
    Inner* orrore2 = static_cast<Inner*>(orrore1);
    delete orrore2;

    ...
}
```

- *static_cast* funziona solo con *void**
- Ma il C++ permette anche questo!!

Casting

- `static_cast`
- `dynamic_cast`
- `reinterpret_cast`
- `const_cast`

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

209

209

Templates

Una veloce introduzione al C++

Giovanni Squillero
squillero@polito.it

210

Template

- L'utilizzo di template permette di «parametrizzare» classi e funzioni (*famiglia* di classi, *famiglia* di funzioni)

```
std::vector<int> vettore{1, 2, 3, 4, 5};
```



- Un vettore di interi
 - Leggibilità
 - Riutilizzabilità

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

211

211

Templates



```
#include <typeinfo>

template<typename T>
class Contenitore {
    T val_;
public:
    Contenitore(T val) : val_{val} { }
    T val() { return val_; }
    void val(T v) { val_ = v; }
};

int main() {
    Contenitore<int> ci{42};
    std::cout << ci.val() << ":" << typeid(ci.val()).name() << std::endl;
    Contenitore<double> cf{4.2};
    std::cout << cf.val() << ":" << typeid(cf.val()).name() << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

212

212

Templates

```
#include <typeinfo>

template<typename T>
class Contenitore {
    T val_;
public:
    Contenitore(T val) : val_{val} { }
    T val() { return val_; }
    void val(T v) { val_ = v; }
};

int main() {
    Contenitore<int> ci{42};
    std::cout << ci.val() << ":" << typeid(ci.val()).name() << std::endl;
    Contenitore<double> cf{4.2};
    std::cout << cf.val() << ":" << typeid(cf.val()).name() << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

213

Funzioni e deduzione

```
template<typename T>
void f(T x) { std::cout << x << " in " << sizeof (T) << " bytes" << std::endl; }

int main() {
    f(4);
    f(4.0);
    f('4');
    f<double>(4);
}
```

4 in 4 bytes
4 in 8 bytes
4 in 1 bytes
4 in 8 bytes

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

214

Parametrizzare una funzione

- È possibile parametrizzare una funzione utilizzando i template

```
template<int val>
bool soglia(int x) { return x > val; }

int main(void) {
    std::cout << soglia<100>(43) << std::endl;
```

Template senza argomenti

```
template<>
struct Foo<int> {
    ...
}
```

Generic programming

- Nota: anche la keyword `auto` può essere utilizzata per ottenere funzioni generiche

```
auto somma(auto x, auto y) { return x + y; };
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

217

217

Standard Library

Una veloce introduzione al C++

Giovanni Squillero
squillero@polito.it

218

S

Una

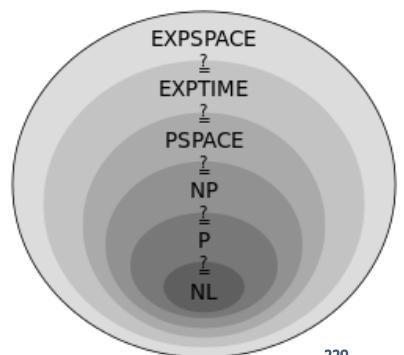
But first...

Giovanni Squillero
squillero@polito.it

219

Complessità Computazionale

- Complessità asintotica
- Notazione O
 - $O(1)$
 - $O(\log n)$
 - $O(n)$
 - $O(n \log n)$
 - $O(n^x)$
 - $O(e^n)$
- Compelssità vs. Tempo di esecuzione



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

220

220

Tempo di esecuzione



```
#include <iostream>
#include <chrono>

int main() {
    auto start = std::chrono::high_resolution_clock::now();
    // calcoli...
    auto end = std::chrono::high_resolution_clock::now();

    std::cout << std::chrono::duration<double, std::milli>(end - start).count()
        << "ms" << std::endl;

    return 0;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

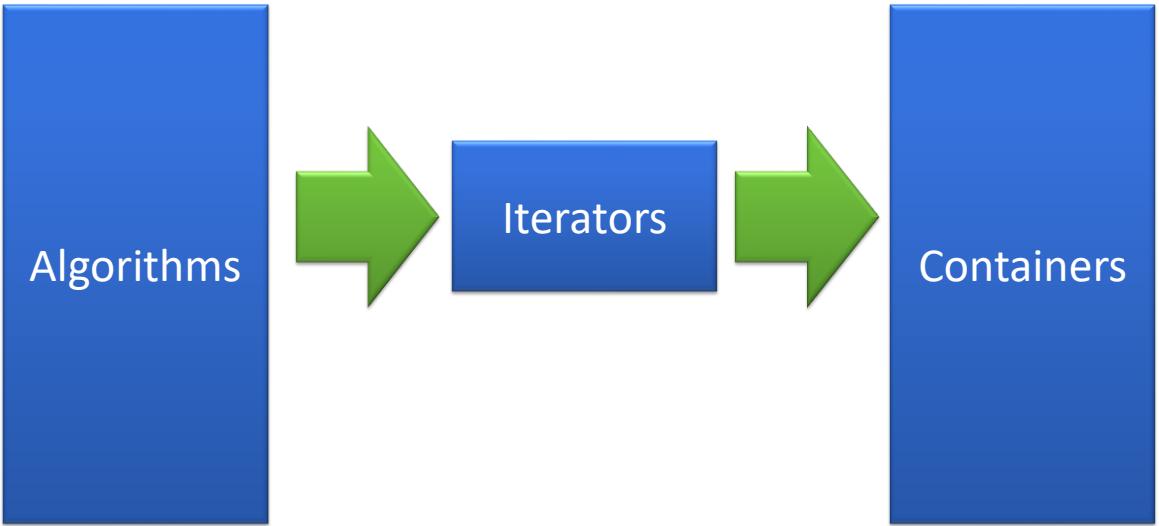
221

Standard Library

Una veloce introduzione al C++

Giovanni Squillero
squillero@polito.it

Standard Template Library



223

C++ STL



- *Standard*
- *Efficiente (ottimizzata)*
- *Accurata*
- *Riutilizzo del codice (non inventare di nuovo la ruota)*

224

Iterators



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

int main() {
    std::vector<int> vec;

    vec.push_back(23);
    vec.push_back(10);

    std::vector<int>::iterator begin = vec.begin();
    std::vector<int>::iterator end = vec.end();

    for(std::vector<int>::iterator i = begin; i != end; ++i)
        std::cout << *i << " ";
    std::cout << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

225

Iterators

- Half-open

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

226

Algorithms



```
#include <algorithm>

int main() {
    std::vector<int> vec;

    for(auto t = 0; t < 20; ++t)
        vec.push_back(std::rand() % 100);

    for(auto& i: vec) std::cout << i << " ";
    std::cout << std::endl;

    std::sort(vec.begin(), vec.end());

    for(auto& i: vec) std::cout << i << " ";
    std::cout << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

227

227

Algorithms



```
#include <algorithm>

int main() {
    std::vector<int> vec;

    for(auto t = 0; t < 20; ++t)
        vec.push_back(std::rand() % 100);

    for(auto& i: vec) std::cout << i << " ";
    std::cout << std::endl;

    std::sort(vec.begin(), vec.end());

    for(auto& i: vec) std::cout << i << " ";
    std::cout << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

228

228

range-for loop

```
for(auto& a: vec) { ... }
```

- «Zucchero sintattico»

```
e_ = vec.end();
for(auto a_ = vec.begin(); a_ != e_; ++a_ )
{
    auto& a = *a_;
    ...
}
```

STL Containers

- Sequenze (array e linked list)
 - vector, deque, list, forward list, array
- Contenitori associativi (alberi)
 - map, multimap, set, multiset
- Contenitori non-ordinati (tabelle di hash)
 - unordered map, unordered multimap, unordered set, unordered multiset

STL Headers



```
#include <vector>
#include <deque>
#include <list>
#include <array>
#include <set>
#include <map>
#include <unordered_set>
#include <unordered_map>
#include <iterator>
#include <algorithm>
#include <numeric>
#include <functional>
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

231

231

Elementi comuni

- Alcune funzioni comuni a tutti i container
 - Copy constructor
 - empty
 - size
 - clear
 - swap
- Nota: nessuna perdita di performance per polimorfismo o astrazione

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

232

232

Vector

- Memoria allocata dinamicamente e contigua
- Inserimento:
 - `push_back(elem)`
- Random access:
 - `at(i), [i]`
- Complessità:
 - Inserire/rimuovere un elemento alla fine: $O(1)$
 - Inserire/rimuovere un elemento: $O(n)$
 - Cercare un elemento: $O(n)$ 

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

233

233

Deque

- Inserimento:
 - `push_front(elem), push_back(elem)`
- Random access:
 - `at(i), [i]`
- Complessità:
 - Inserire/rimuovere un elemento alla fine o all'inizio: $O(1)$
 - Inserire/rimuovere un elemento: $O(n)$
 - Cercare un elemento: $O(n)$ 

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

234

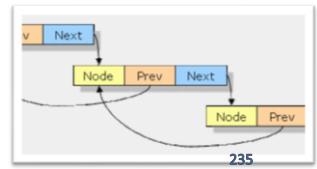
234

List

- Inserimento:
 - `push_front(elem)`, `push_back(elem)`,
`insert(iter, elem)`
- Cancellazione:
 - `erase(iter)`
- Complessità:
 - Inserire/rimuovere un elemento: $O(1)$
 - Cancellare un elemento: $O(1)$
 - Cercare un elemento: $O(n)$ ←

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp



235

Forward List

- Inserimento:
 - `push_front(elem)`, `insert(iter, elem)`
- Cancellazione:
 - `erase(iter)`
- L'iteratore non può muoversi all'indietro (`--iter`)
- Complessità:
 - Inserire/rimuovere un elemento: $O(1)$
 - Cancellare un elemento: $O(1)$
 - Cercare un elemento: $O(n)$ ←

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

236

Array Container

- Wrapper per il «tradizionale» array del C

```
std::array<int, 3> data = {1, 2, 3};
```

- Permette di usare
 - begin, end, size, swap
- Overhead minimo

Suggerimenti (ricordate?)

- Usate `std::vector` per tutti i problemi complicati (e.g., numero di elementi variabile)
- Usate `std::array` nei casi non-banali (se serve particolare efficienza)
- Usate i «vecchi» array il meno possibile, e in casi molto speciali



List vs. Deque vs. Vector

- La memoria di una deque non è contigua
- Efficienza vs. Complessità
- Cache, prefetch

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

239

239

splice

- Rimuove una porzione da una lista e la inserisce in una precisa posizione in un'altra lista

```
lista1.splice(iterator1, lista2, iterator2_inizio, iterator2_fine)
```



- L'operazione è O(1)

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

240

240

Coppie di valori

- `pair` è una struttura che permette di unire due valori eterogenei in una singola unità
 - standard
 - veloce
 - minimo overhead

```
#include <utility>

std::pair<int, bool> p1;
p1.first = 42;
p1.second = false;
auto p2 = std::make_pair(42, true);
```

Set

- Inserimento:
 - `insert(elem)`, `insert(iter, elem)`
- Ricerca:
 - `find(elem)`
- Cancellazione:
 - `erase(iter)`, `erase(elem)`
- Complessità:
 - Inserire/rimuovere un elemento: $O(\log n)$
 - Cercare un elemento: $O(\log n)$

Set

- Inserimento:

- `insert(elem)`, `insert(iter, elem)`

- Ricerca:

```
std::pair<iterator,bool> insert(value_type&& value);
```

suggerimento

- Cancellazione:

- `erase(iter)`, `erase(elem)`

- Complessità:

- Inserire/rimuovere un elemento: $O(\log n)$

- Cercare un elemento: $O(\log n)$

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

243

243

Set (esempio)



```
#include <iostream>
#include <set>

int main() {
    std::set<int> data;

    for(auto t=0; t<50; ++t) {
        auto n = std::rand() % 100;
        auto val = data.insert(n);
        if(!val.second) std::cout << "Collision on " << n << std::endl;
    }

    for(auto& i: data)
        std::cout << i << " ";
    std::cout << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

244

244

Multiset

- Inserimento:
 - `insert(elem)`, `insert(iter, elem)`
- Ricerca:
 - `find(elem)`
- Cancellazione:
 - `erase(iter)`, `erase(elem)`
- Complessità:
 - Inserire/rimuovere un elemento: $O(\log n)$
 - Cercare un elemento: $O(\log n)$

Multiset vs. Vector

- Inserisco i dati in una struttura ad albero
 - Complessità $O(n \log n)$
- Inserisco i dati in un vettore e poi lo ordino alla fine
 - Complessità $O(1) + O(n \log n)$
- Cosa conviene fare?

Multiset vs. Vector



```
constexpr int NUM_TEST = 10000000;

int main() {
    std::multiset<int> data1;

    auto start = std::chrono::high_resolution_clock::now();
    for(auto t=0; t<NUM_TEST; ++t) {
        auto n = std::rand();
        data1.insert(n);
    }
    auto end = std::chrono::high_resolution_clock::now();

    std::cout << std::chrono::duration<double, std::milli>(end - start).count()
          << "ms" << std::endl;
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

247

Multiset vs. Vector



```
std::vector<int> data2;

start = std::chrono::high_resolution_clock::now();
for(auto t=0; t<NUM_TEST; ++t) {
    auto n = std::rand();
    data2.push_back(n);
}
std::sort(data2.begin(), data2.end());
end = std::chrono::high_resolution_clock::now();

std::cout << std::chrono::duration<double, std::milli>(end - start).count()
      << "ms" << std::endl;
...
}
```

15244.1ms
3478.34ms

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

248



Multiset vs. Vector

```

data2.reserve(NUM_TEST);

std::vector<int> data2;

start = std::chrono::high_resolution_clock::now();
for(auto t=0; t<NUM_TEST; ++t) {
    auto n = std::rand();
    data2.push_back(n);
}
std::sort(data2.begin(), data2.end());
end = std::chrono::high_resolution_clock::now();

std::cout << std::chrono::duration<double, std::milli>(end - start).count()
     << "ms" << std::endl;

...
}

```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

249

249

Set/Multiset

- Ricerca veloce $O(\log n)$
- Attraversamento lento (località)
- No accesso casuale

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

250

250

Set/Multiset

- Gli elementi non possono essere modificati

```
std::vector<int> data1;
data1.push_back(23);
auto iter1 = data1.begin();
*iter1 = 10;      // OK

std::multiset<int> data2;
data2.insert(23);
auto iter2 = data2.begin();
*iter2 = 10;      // error: assignment of read-only location
```

Set/Multiset

- Controllare se la ricerca è veramente più veloce

Set di Classi

- È possibile avere set/multiset di classi
- È necessario definire l'operatore <

```
struct Mynum {  
    int v;  
};  
  
bool operator<(Mynum const &a, Mynum const &b) {  
    return a.v > b.v;  
}
```

Set di Classi

```
int main() {  
    std::set<Mynum> data;  
  
    data.insert(Mynum{23});  
    data.insert(Mynum{10});  
    data.insert(Mynum{18});  
    data.insert(Mynum{5});  
  
    for(auto& i: data)  
        std::cout << i.v << " ";  
    std::cout << std::endl;  
}
```

23 18 10 5

pair<Key, Value>

Map/Multimap

- Inserimento:
 - insert(pair), insert(iter, pair)
- Ricerca:
 - find(key)
- Cancellazione:
 - erase(iter), erase(key)
- Complessità:
 - Inserire/rimuovere un elemento: $O(\log n)$
 - Cercare un elemento: $O(\log n)$

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

255

255

Map/Multimap



```
#include <map>

int main() {
    std::map<std::string, int> data;

    for(auto s: {"foo", "bar", "baz", "gargle", "bar"} ) {
        auto r = data.insert(std::make_pair(s, 0));
        ++r.first->second;
    }

    for(auto& i: data)
        std::cout << i.first << ":" << i.second << " ";
    std::cout << std::endl;
}
```

bar:2 baz:1 foo:1 gargle:1

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

256

256

Quiz

- Spiegare cosa significa

```
++r.first->second;
```

squillero@polito.it

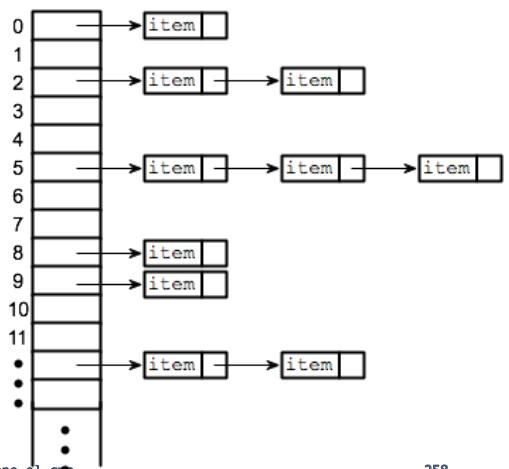
https://github.com/squillero/introduzione_al_cpp

257

257

C++11 Unordered Containers

- Hash table
 - Hash function
 - Buckets
 - Entries
- Operazioni O(1)



squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

258

258

Unordered set

- Inserimento:

- `insert(elem)`, `insert(altro.itri, altro.itrf)`

- Ricerca:

- `find(elem)`

- Cancellazione:

- `erase(iter)`, `erase(elem)`

- Complessità:

- Inserire/rimuovere un elemento: $O(1)$

- Cercare un elemento: $O(1)$

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

259

259

Unordered set



```
#include <unordered_set>

int main() {
    std::unordered_set<std::string> data = { "foo", "bar", "baz", "gargle" };

    for(auto& i: data)
        std::cout << i << " ";
    std::cout << std::endl;

    std::cout << "Load factor: " << data.load_factor() << std::endl;
    std::cout << "Total buckets: " << data.bucket_count() << std::endl;
    std::cout << "Foo's bucket: " << data.bucket("foo") << std::endl;
    std::cout << "Foo's bucket size: " << data.bucket_size(data.bucket("foo"))
                  << std::endl;
}
```

```
gargle baz bar foo
Load factor: 0.8
Total buckets: 5
Foo's bucket: 4
Foo's bucket size: 1
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

260

Unordered multiset

- Inserimento:
 - `insert(elem)`, `insert(altro.itri, altro.itrf)`
- Ricerca:
 - `find(elem)`, `count(elem)`
- Cancellazione:
 - `erase(iter)`, `erase(elem)`
- Complessità:
 - Inserire/rimuovere un elemento: $O(1)$
 - Cercare un elemento: $O(1)$

Unordered Map/Unordered Multimap

- Come map/multimap, ma non ordinate

Array associativi

- Possono essere realizzata con map/unordered_map
- Operatore [] e operatore at()

```
int main(void) {
    std::unordered_map<std::string, int> freq;

    for(auto s: {"foo", "bar", "baz", "gargle", "bar"} ) {
        ++freq[s];
    }

    for(auto& k: freq)
        std::cout << k.first << ":" << k.second << " ";
    std::cout << std::endl;
}

return 0;
}
```

```
gargle:1 baz:1 bar:2 foo:1
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

263

263

Hash di Classi

- Per usare classi come chiavi di unordered_set/map è necessario definire una funzione di hash

```
struct Test {
    int num;
    bool operator==(const Test& o) const {
        return num == o.num;
    }
};

namespace std
{
    template<>
    struct hash<Test> {
        size_t operator()(const Test & t) const {
            return hash<int>()(t.num);
        }
    };
}
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

264

264

#include <functional>

```
template<> struct hash<bool>;
template<> struct hash<char>;
template<> struct hash<signed char>;
template<> struct hash<unsigned char>;
template<> struct hash<char16_t>;
template<> struct hash<char32_t>;
template<> struct hash<wchar_t>;
template<> struct hash<short>;
template<> struct hash<unsigned short>;
template<> struct hash<int>;
template<> struct hash<unsigned int>;
template<> struct hash<long>;
template<> struct hash<long long>;
template<> struct hash<unsigned long>;
template<> struct hash<unsigned long long>;
template<> struct hash<float>;
template<> struct hash<double>;
template<> struct hash<long double>;
template< class T > struct hash<T*>;
```

```
std::hash<std::string>
std::hash<std::u16string>
std::hash<std::u32string>
std::hash<std::wstring>
```

```
std::hash<std::error_code>
std::hash<std::bitset>
std::hash<std::unique_ptr>
std::hash<std::shared_ptr>
std::hash<std::type_index>
std::hash<std::vector<bool>>
std::hash<std::thread::id>
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

265

265

Container Adoptor

- stack
 - push, pop, top
- queue
 - push, pop, front, back
- priority queue
 - push, pop, top

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

266

266

Tassonomia alternativa

- Basati su array
 - vector, deque
 - Le operazioni sul contenitore possono invalidare puntatori, iteratori, riferimenti
- Basati su nodi
 - I puntatori ai nodi rimangano validi

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

267

267

Trovare l'errore

- `unordered_set` e `vector`

```
std::unordered_set<Frutta *> cesto;

cesto.insert(new Mela{"golden"});
cesto.insert(new Mela{"annurca"});
cesto.insert(new Pera{"ruset"});
cesto.insert(new Pera{"mora"});

for(auto a : cesto)
    a->tipo();

auto saved = cesto.begin();
std::vector<Frutta*> vettorev;
vettorev.push_back(saved);
```

squillero@polito.it

https://github.com/squillero/introduzione_al_cpp

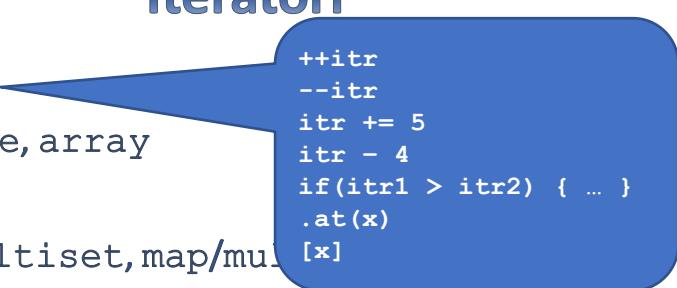
268

268

Iteratori

- Random access
 - `vector`, `deque`, `array`
- Bidirezionali
 - `list`, `set/multiset`, `map/multimap`
- Unidirezionali
 - `forward_list`

Iteratori

- Random access
 - `vector`, `deque`, `array`
 - Bidirezionali
 - `list`, `set/multiset`, `map/multimap`
 - Unidirezionali
 - `forward_list`
- 
- ```
++itr
--itr
itr += 5
itr - 4
if(itr1 > itr2) { ... }
.at(x)
[x]
```

# Iteratori

- Random access
  - vector, deque, 
- Bidirezionali
  - list, set/multiset, map/multimap
- Unidirezionali
  - forward\_list

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

271

271

# Iteratori

- Random access
  - vector, deque, array
- Bidirezionali
  - list, set/multiset, map/multimap 
- Unidirezionali
  - forward\_list

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

272

272

# Iteratori

- Input iterator
  - `int x = *itr;`
- Output iterator
  - `*itr = 42;`
- Const iterator

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

273

273

# Funzioni per iteratori

- Utilizzabili con tutti gli iteratori
  - `advance`
  - `distance`

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

274

274

## Iteratori speciali

- Insert iterator
  - `back_insert_operator`
  - `front_insert_operator`
- Reverse iterator
- Move iterator (C++11)

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

275

275

## Iteratori e puntatori

- I puntatori (anche grezzi) sono iteratori

```
int array[4] = { 23, 10, 18, 5 };
std::sort(array, array + 4);
```

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

276

276

# Algoritmi

- `sort(inizio, fine)`
- `reverse(inizio, fine)`
- `copy(srg_inizio, srg_fine, dest)`
  - deve esserci spazio sufficiente nella destinazione
  - oppure usare un insert iterator (`std::back_inserter()`)
  - Ancora meglio usare la funzione membro  
`insert(dest, srg_inizio, srg_fine)`

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

277

277

# Functors

- Un *functor* o «oggetto funzione» è una classe che contiene l'operatore chiamata-a-funzione

```
class Soglia {
 int val;
public:
 Soglia() : val{0} { };
 Soglia(int x) : val{x} { };
 bool operator () (int x) { return x > val; }
};

int main(void) {
 Soglia sufficiente{18};
 Soglia ottimo{28};
 std::cout << sufficiente(20) << ", " << ottimo(20) << std::endl;
 std::cout << Soglia(199)(203) << std::endl;
 ...
}
```

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

278

278

## Alternativa con Template

- È possibile parametrizzare una funzione utilizzando i template

```
template<int val>
bool soglia(int x) { return x > val; }

int main(void) {
 std::cout << soglia<100>(43) << std::endl;
```

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

279

279

## Call wrappers

- È possibile costruire delle funzioni, congelando alcuni parametri

```
bool compreso(int x, int min, int max) {
 return x < max && x > min;
}

int main(void) {
 auto f = std::bind(compreso, std::placeholders::_1, 10, 20);
 std::cout << f(15) << std::endl;
```

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

280

280

## Funzioni disponibili

- Definiti in <functional>

```
plus<T> f(a, b);
minus<T> f(a, b);
multiplies<T> f(a, b);
divides<T> f(a, b);
modulus<T> f(a, b);
equal_to<T> f(a, b);
not_equal_to<T> f(a, b);
greater<T> f(a, b);
greater_equal<T> f(a, b);
less<T> f(a, b);
less_equal<T> f(a, b);
logical_and<T> f(a, b);
logical_or<T> f(a, b);

negate<T> f(a);
logical_not<T> f(a);
```

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

281

281

## Funzioni lambda

- È possibile costruire delle funzioni, congelando alcuni parametri

```
int main(void) {
 auto f = [] (int x) { return x < 20 && x > 10; };
 std::cout << f(15) << std::endl;
 ...
}
```

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

282

282

## Algoritmi

- `find_if(inizio, fine, condizione)`
- `for_each(inizio, fine, funzione)`
- `for_each(inizio, fine, funzione)`

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

283

283

## Algoritmi

- `count(inizio, fine, valore)`
- `count_if(inizio, fine, condizione)`
- `min_element(inizio, fine[, confronto])`
- `max_element(inizio, fine[, confronto])`
- `minmax_element(inizio, fine[, confronto])`

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

284

284

## Algoritmi (ricerca lineare)

- `find(inizio, fine, valore)`
- `find_if(inizio, fine, condizione)`
- `find_if_not(inizio, fine, condizione)`
- `search_n(inizio, fine, num, valore)`
- `search(inizio, fine, pat_inizio, pat_fine)`
- `find_end(inizio, fine,  
              pat_inizio, pat_fine)`

## Algoritmi (ricerca lineare)

- `find_first_of(inizio, fine,  
                  valori_inizio, valori_fine  
                  [, condizione])`
- `adjacent_find(inizio, fine[, condizione])`

## Confronto

- `equal(inizio, fine, confronto)`
- `is_permutation(inizio, fine, confronto)`
- `mismatch(inizio1, finel, inizio2)`
- `lexicographical_compare(inizio1, finel,  
inizio2, fine2)`
- Tutte le funzioni hanno una forma generale

## Attributi

- `is_sorted(inizio, fine)`
- `is_sorted_until(inizio, fine)`
- `is_partitioned(inizio, fine, condizione)`
- `is_heap(inizio, fine)`
- `is_heap_until(inizio, fine)`
- Tutte le funzioni hanno una forma generale

## Attributi

- `all_of(inizio, fine, condizione)`
- `any_of(inizio, fine, condizione)`
- `none_of(inizio, fine, condizione)`

squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

289

289

## Esempio

```
int main(void) {
 std::unordered_set<Test> data; hash personalizzata

 data.insert(Test{10});
 data.insert(Test{23});
 data.insert(Test{18});
 data.insert(Test{5});

 std::for_each(data.cbegin(), data.cend(),
 [] (auto x) { std::cout << x.num << std::endl; });

 return 0;
}
```

algoritmo

iteratori costanti

funzione lambda



squillero@polito.it

[https://github.com/squillero/introduzione\\_al\\_cpp](https://github.com/squillero/introduzione_al_cpp)

290

290