

Luiss
Libera Università Internazionale
degli Studi Sociali Guido Carli

Corso di preparazione per la selezione territoriale delle Olimpiadi di Informatica

Lezione 1. Introduzione al C++

Blerina Sinaimeri

27 febbraio 2023



Programma di oggi

1. Introduzione alla piattaforma, installazione IDE
2. Include, namespace, input/output, variabili
3. Tipi e struct
4. Parsing dell'input, `std::string`
- 5. Esercizi olimpiadi**
6. Booleani, if, else
7. Cicli for e while
8. Accenni di complessità computazionale
- 9. Esercizi olimpiadi**
10. `std::vector` & sorting con STL
11. Insiemi e dizionari (`std::set`, `std::unordered_map`)
12. `std::stack`, `std::queue`
- 13. Esercizi olimpiadi**




Piattaforme di allenamento



Piattaforma di allenamento

- Piattaforma ufficiale olimpiadi italiane: <https://training.olinfo.it/>

Scegli il tuo
“linguaggio”

	C++		JavaScript
	Java/C#		PHP(Without MySQL)
	Ruby		Pascal
	Perl		Lisp
	Visual Basic		Haskell
	Python		C

@左耳朵耗子
weibo.com/hael

Integrated Development Environment (IDE): Dev-C++



IDE: Dev-C++

- Un IDE (integrated development environment) è un programma che ci semplifica il compito di scrivere codice, compilarlo ed eseguirlo.
- Useremo Dev-C++:
<https://sourceforge.net/projects/orwellddevcpp/>
- Primo compito di oggi: scaricate e installate Dev-C++.

Configurare il compilatore: standard C++17

- Il linguaggio C++ è in continua evoluzione:
 - Prima implementazione: 1985
 - Standards: C++98, C++03, C++11, C++14, C++20 (corrente versione stabile)
- Noi scriveremo codice secondo lo standard C++17

Configurare il compilatore: standard C++17

- Il linguaggio C++ è in continua evoluzione:
 - Prima implementazione: 1985
 - Standards: C++98, C++03, C++11, C++14, C++20 (corrente versione stabile)
- Noi scriveremo codice secondo lo standard C++17
- Secondo compito di oggi: configurate Dev-C++ per C++17:
 - Tools → Compiler Options → General
 - Selezionare «Add the following commands when calling the compiler»
 - Aggiungere nel riquadro la linea «-std=c++17»

Reminder per dopo: come scrivere ed eseguire codice

1. File → new → source file.
2. Scrivere il codice nel riquadro.
3. File → Save As ... salvate il file con il nome che volete.
4. Execute → Compile & Run.

Include, namespace, main, cout, commenti



Codice minimale: hello world (copiatelo ed eseguitelo!)

```
#include <iostream>
using namespace std;
int main(){
    cout << "hello world" << endl;
    //questo è un commento
    return 0;
}
```

Ora vediamo il significato di questa sintassi, passo per passo.

Include

- Nel nostro codice possiamo includere librerie contenenti funzioni/oggetti utili.
- Per includere una libreria: `#include <libreria>`
- Nel nostro esempio precedente, includiamo una libreria per fare input/output (`iostream`)

namespace

- Al suo interno, la libreria racchiude le definizioni in diversi **namespace** (spazi dei nomi).
- Per usare un oggetto `pip` definito nel namespace `spazio`, dovrò scriverlo come `spazio::pip`

namespace

- Al suo interno, la libreria racchiude le definizioni in diversi **namespace** (spazi dei nomi).
- Per usare un oggetto `pip` definito nel namespace `spazio`, dovrò scriverlo come `spazio::pip`
- In alternativa (preferibile), scriviamo in alto `using namespace spazio`; in questo modo non dobbiamo preoccuparci di scrivere ogni volta `spazio::` prima degli oggetti.

main

- `main()` è la funzione che viene eseguita all'avvio del nostro programma.
- Scriveremo il nostro codice tra le parentesi graffe di `main()`.

```
#include <iostream>

using namespace std;

int main(){

    cout << "hello" << endl;

    return 0;

}
```


cout

- Infine, «cout» è lo stream di output: tutto quello che spediamo verso di lui con l'operatore «<<» viene stampato a schermo.
- «endl» ci manda a capo.

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    cout << "hello" << endl;
```

```
    return 0;
```

```
}
```



Esercizio: Usando una sola volta «cout», scrivere un programma che stampa a schermo il proprio nome e cognome su due linee diverse.

Variabili, Input/output da/verso file



cin, cout da/verso file

- «cin» è lo stream di input (default: da tastiera).
- In questo corso ci interesserà salvare i risultati su file (non a schermo)
- Analogamente, vogliamo leggere il nostro input da un file (non da tastiera).
- Prossime slide: **redirezione** di cin/cout da/verso file.

cin, cout da/verso file

- Alle olimpiadi si usano questi file di input/output:
 - input.txt
 - output.txt
- Create un file di testo chiamato «input» nella cartella contenente il vostro codice (attenzione: windows aggiunge «.txt» automaticamente).
- Salvateci dentro il vostro nome, cognome, età (in anni), separati da uno spazio.
- **Esempio:** Mario Rossi 16

Lettura file → **variabili** e scrittura **variabili** → file.

```
#include <iostream>

using namespace std;

int main(){
    freopen("input.txt","r",stdin); //cin legge da file
    freopen("output.txt","w",stdout); //cout viene redirezionato su file
    string nome,cognome; //creo due variabili nome, cognome di tipo string
    int eta; //creo una variabile età di tipo int
    cin >> nome >> cognome >> eta; //leggo nome, cognome, età nelle tre variabili
    cout << nome; // salvo su file il mio nome.
}
```

Tipi e struct



Tipi interi

- Fate molta attenzione ai **tipi** interi.
- Non c'è solo un tipo di numero intero! Diversi range/numero di bit:
 - int
 - unsigned int
 - long int
 - unsigned long int
 - short int
 - ...

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

Tipi interi

...

Problema: i range dei tipi qui sopra dipendono dall'architettura del processore.

- Personalmente, non mi ricordo mai quanti bit abbia un long int/long long int 😊
- Soluzione: consiglio di usare tipi interi **architecture-independent** (prossima slide)

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

Tipi interi architecture-independent

Nome tipo intero	bit	da	a
int8_t	8	$-2^7 = -128$	$2^7 - 1 = 127$
uint8_t	8	0	$2^8 - 1 = 255$
int16_t	16	$-2^{15} = -32\,768$	$2^{15} - 1 = 32\,767$
uint16_t	16	0	$2^{16} - 1 = 65\,535$
int32_t	32	$-2^{31} = -2\,147\,483\,648$	$2^{31} - 1 = 2\,147\,483\,647$
uint32_t	32	0	$2^{32} - 1 = 4\,294\,967\,295$
int64_t	64	$-2^{63} = -9\,223\,372\,036\,854\,775\,808$	$2^{63} - 1 = 9\,223\,372\,036\,854\,775\,807$
uint64_t	64	0	$2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615$

Overflow e underflow

Attenzione! Cosa stampa questo codice? Perché?

```
uint32_t x = 0-1;
```

```
cout << x;
```

Fate molta attenzione a underflow/overflow di variabili intere: **fonte di innumerevoli bug**

Esercizi olimpiadi



Esercizio 1

Possiamo finalmente risolvere dei semplici esercizi OII!

- Exam room(examroom): [ois_examroom](#)

Tipi e struct

- Abbiamo visto come creare variabili di **tipo** int (numero intero) e string (stringa).
- Possiamo raggruppare variabili e formare tipi più complessi usando struct.

struct

```
#include <iostream>
using namespace std;
struct persona{
    string nome;
    string cognome;
    int eta;
};
int main(){
    freopen("input.txt","r",stdin);
    freopen("output.txt","w",stdout);
    persona P;
    cin >> P.nome >> P.cognome >> P.eta;
    cout << P.nome << " " << P.cognome << " ha " << P.eta << " anni";
}
```

Esercizio



1. Salvare in «input.txt» i nomi, cognomi, età di tre persone:
 - I dati della stessa persona sono separati da spazio.
 - I dati delle tre persone stanno su tre linee diverse.
2. Leggere i dati delle tre persone in tre variabili di tipo persona.
3. Salvare su file la media delle età (approssimata per difetto) delle tre persone.
4. Suggerimento 1: tra variabili int possiamo usare i soliti operatori aritmetici +, -, *, /
5. Suggerimento 2: la lettura da stream «cin >> ... >> ...» spezza il file in token usando come separatori spazio, newline, tab.

Parsing dell'input

std::string, std::istream



Standard Library: STL

- STL è la **standard library** C++.
- È un insieme di librerie che possiamo includere nei nostri programmi.
- Mette a disposizione algoritmi e strutture dati che ci torneranno molto utili nella risoluzione dei nostri problemi.

`std::string, std::istringstream`

- Abbiamo già fatto la conoscenza di alcuni oggetti STL: **`std::string`**, **`std::cin`**, **`std::cout`**, ...
- Approfondiamo la conoscenza di **`std::string`** e del relativo input stream **`std::istringstream`**.
- Fondamentali per un semplice parsing dell'input!

std::string, std::istringstream

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

<http://www.cplusplus.com/reference/sstream/istringstream/>

When you start coding in a new language without reading the documentation



`std::string, std::istringstream`

- Prima funzionalità utilissima: **tokenization**.
- Ci permette di leggere token da `cin`, usando come separatore un qualunque carattere (non solo spazi, tab, newline)
- Scrivete nel vostro file di input una linea «nome;cognome;età»

getline(): spezziamo la stringa usando un separatore qualsiasi

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

```
int main(){
    freopen("input.txt","r",stdin);
    freopen("output.txt","w",stdout);

    string nome, cognome, etas;
    getline(cin,nome,';'); //leggo le tre variabili dallo stream usando come separatore ;
    getline(cin,cognome,';');
    getline(cin,etas,';');

    int eta = stoi(etas); //convertito etas in int
    cout << nome << " " << cognome << " " << eta << endl;
}
```



- Usando il blocco note, scrivere in «input.txt» una linea nel formato gg;hh:mm:ss;ev. **Per esempio:**

10 Marzo 2022;16:00:00;lezione C++

- Scrivere un programma che legga «input.txt» e scriva in «output.txt» la frase «Il giorno gg alle ore hh e mm minuti inizia l'evento ev». **Per esempio:**

Il giorno 10 Marzo 2022 alle ore 16 e 30 minuti inizia l'evento lezione C++

std::string

Altre funzioni utili (non le uniche! Leggete il manuale):

```
string s = "impariamo";  
cout << s[2] << endl; // "p"  
cout << s.substr(1,4) << endl; // "mpar"  
cout << s.length() << endl; // 9  
cout << (s + " a usare le stringhe") << endl; //concatenazione
```


bool, if/else



bool

- Un nuovo tipo di dato: bool.
- Una variabile bool può assumere i valori true (1) o false (0).
- Possiamo creare variabili booleane usando operatori di confronto: `bool x = (a < 7);` (ma non solo, come vedremo)
- Possiamo combinare variabili booleane con gli operatori not, and e or: `bool x = not(a < 13 and a > 4);`

bool

Esercizio: Cosa restituisce questo pezzo di codice?

```
int a = 3;  
bool x = (a<5);  
bool y = (a>=1);  
cout << not(x and y);
```

If, else

- I booleani possono essere usati come condizioni per eseguire pezzi di codice in modo condizionale.

```
int a;  
cin >> a;  
if(a%3 == 0){  
    cout << a << " e' divisibile per 3";  
  
}else if(a % 2 == 0){  
    cout << a << " e' divisibile per 2 e non per 3";  
  
}else{  
    cout << a << " non e' divisibile ne' per 2 ne' per 3";  
  
}
```

Cicli for e while



Cicli for e while

- Un ciclo **for** ci permette di eseguire un blocco di istruzioni per un numero **predefinito** di volte.

```
#include <iostream>
using namespace std;
int main(){

    int n = 10;

    for(int i=0;i<n;++i){

        //blocco di istruzioni da ripetere
        cout << i << endl;

    }

}
```

Cicli for e while

Regole generali:

- Sebbene sia possibile *modificare* il contatore `i` dentro al ciclo `for`, evitate di farlo (per questo c'è il ciclo `while`)
- Potete (spesso, dovete) *usare* il contatore `i` dentro al corpo del ciclo `for`
- Meglio dichiarare il contatore `i` dentro il costrutto `for` (anche se non è necessario)
- La funzione di aggiornamento del contatore e la condizione di terminazione possono essere arbitrarie. Esempi:
 - `for(int i=0; i < 1000; i *= 2)`
 - `for(int i=k; i > 1000 & i < 6000; i = i*i + 2*i)`

Cicli for e while

- Un ciclo **while** ci permette di eseguire un blocco di istruzioni per un numero di volte deciso a runtime.

```
#include <iostream>
using namespace std;
int main(){

    int i;

    cin >> i;

    while(i>=0){

        cout << "hai inserito il numero positivo " << i << endl;
        cin >> i;

    }

}
```


Cicli for e while

- Sintassi alternativa (ma con una fondamentale differenza): **do...while**

```
#include <iostream>
using namespace std;
int main(){

    int i;

    cin >> i;

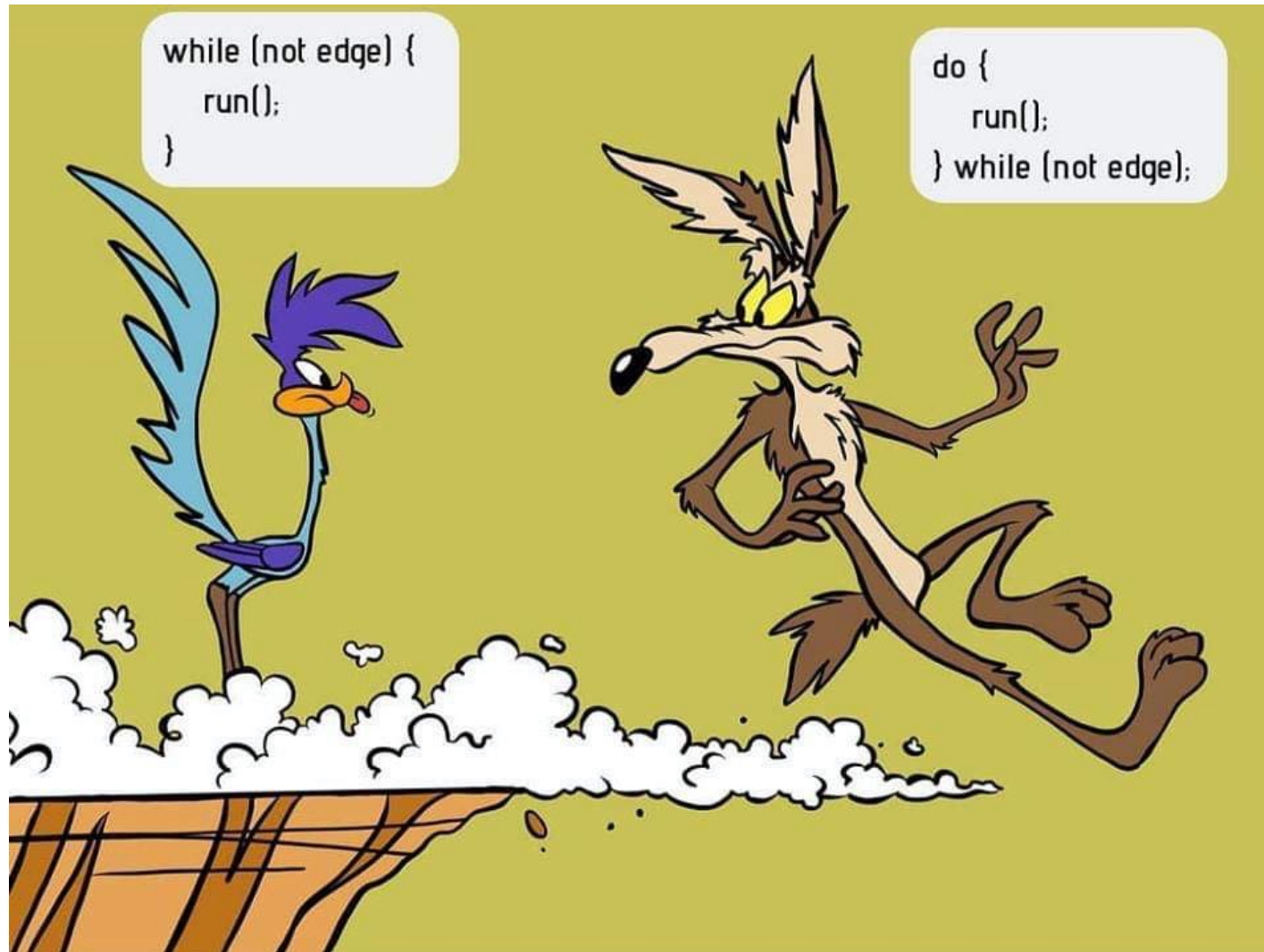
    do{

        cout << "hai inserito il numero " << i << endl;
        cin >> i;

    }while(i>=0);

}
```

Cicli for e while



Esercizi olimpiadi



Esercizio 2

- Bustine di carte (carte): bustine di carte

Complessità computazionale



Accenni di complessità computazionale

- Spesso, alle olimpiadi non basta una soluzione corretta ...
- Ci vuole anche una soluzione efficiente (veloce)!
- Se il programma impiega troppo tempo, la soluzione viene considerata sbagliata.

Accenni di complessità computazionale

Ma come facciamo a sapere quanto veloce è il nostro programma?

- Dobbiamo contare quante operazioni vengono eseguite.
- Ma cos'è un'operazione? Quanto “costa” ogni operazione?

Accenni di complessità computazionale

Usiamo un modello semplificato:

- Diciamo che ogni operazione “di base” (op. aritmetiche, assegnamenti, confronti...) richiede un tempo **costante**.
- Questo significa che il tempo per eseguire quell'operazione non dipende dalla dimensione dell'input.

Accenni di complessità computazionale

Usiamo un modello semplificato:

- Diciamo che ogni operazione “di base” (op. aritmetiche, assegnamenti, confronti...) richiede un tempo **costante**.
- Questo significa che il tempo per eseguire quell'operazione non dipende dalla dimensione dell'input.
- **Esempio:** il mio computer impiega 3 nanosecondi per confrontare due caratteri di una stringa, indipendentemente dalla lunghezza della stringa.
- Un tempo costante viene indicato con il simbolo **$O(1)$**

Notazione O grande

- La notazione O grande “nasconde” le costanti, che non ci interessano troppo.
- Quindi, se un confronto impiega 3 nanosecondi e un assegnamento impiega 5 nanosecondi, dirò che entrambi impiegano tempo $O(1)$: non mi interessa il valore esatto.

Notazione O grande

- La notazione O grande “nasconde” le costanti, che non ci interessano troppo.
- Quindi, se un confronto impiega 3 nanosecondi e un assegnamento impiega 5 nanosecondi, dirò che entrambi impiegano tempo $O(1)$: non mi interessa il valore esatto.
- In generale, indicheremo con la variabile n la dimensione dell’input (lunghezza stringa, numero di interi, ...)
- Leggere/sovrascrivere tutti i caratteri di una stringa lunga n richiede quindi tempo $O(n)$.

Notazione O grande

- Le cose si fanno interessanti quando annido due cicli for ... che complessità ha questo codice?

```
string s;  
cin >> s;  
int n = s.length();  
  
for(int i=0;i<n;++i){// n ripetizioni  
  
    for(int j=i+1;j<n;++j){ //n-i-1 ripetizioni  
  
        if(s[j] < s[i]){// complessità: O(1)  
            char tmp = s[i]; // complessità: O(1)  
            s[i] = s[j]; // complessità: O(1)  
            s[j] = tmp; // complessità: O(1)  
        }  
    }  
}  
  
cout << s; // complessità: O(1)
```

Notazione O grande

La complessità è **quadratica**! $O(n^2)$

- Tra $O(n)$ e $O(n^2)$ ci sono infinite altre complessità: $O(n \log n)$, $O(n \log^2 n)$, $O(n^{1.5})$...
- Ovviamente esistono complessità ancora più grandi, ad esempio $O(2^n)$

Notazione O grande

La complessità è **quadratica**! $O(n^2)$

- Tra $O(n)$ e $O(n^2)$ ci sono infinite altre complessità: $O(n \log n)$, $O(n \log^2 n)$, $O(n^{1.5})$...
- Ovviamente esistono complessità ancora più grandi, ad esempio $O(2^n)$
- **Esempio:** se $n = 2^{30}$ e un'operazione di base impiega 3 nanosecondi, allora:
 - Un algoritmo $O(n)$ termina in alcuni secondi.
 - Un algoritmo $O(n^2)$ termina in alcuni secoli.
 - Un algoritmo $O(2^n)$ richiede più tempo dell'attuale età dell'universo per terminare!

Notazione O grande

La complessità è **quadratica**! $O(n^2)$

- Tra $O(n)$ e $O(n^2)$ ci sono infinite altre complessità: $O(n \log n)$, $O(n \log^2 n)$, $O(n^{1.5})$...
- Ovviamente esistono complessità ancora più grandi, ad esempio $O(2^n)$
- **Esempio:** se $n = 2^{30}$ e un'operazione di base impiega 3 nanosecondi, allora:
 - Un algoritmo $O(n)$ termina in alcuni secondi.
 - Un algoritmo $O(n^2)$ termina in alcuni secoli.
 - Un algoritmo $O(2^n)$ richiede più tempo dell'attuale età dell'universo per terminare!
- Se un problema OI ammette una soluzione $O(n)$, state certi che i creatori del problema avranno settato un timeout che “elimina” le soluzioni $O(n^2)$!
- Spesso una soluzione $O(n \log n)$ sarà sufficiente.

Complessità e STL

- Il manuale STL ci fornisce la complessità di ogni operazione sui suoi oggetti.
- Controllatela bene prima di usare una funzione STL!
- Esempio: siano s , t due stringhe lunghe n . Quelle che seguono sembrano entrambe operazioni di base, ma quanto mi costano? (controllate su <http://www.cplusplus.com/reference/string/string/>)
 - $s[n/2] == t[n/2]$
 - $s.compare(t)$

Complessità e STL

- Il manuale STL ci fornisce la complessità di ogni operazione sui suoi oggetti.
- Controllatela bene prima di usare una funzione STL!
- Esempio: siano s , t due stringhe lunghe n . Quelle che seguono sembrano entrambe operazioni di base, ma quanto mi costano? (controllate su <http://www.cplusplus.com/reference/string/string/>)
 - $s[n/2] == t[n/2]$ **$O(1)$**
 - $s.compare(t)$ **$O(n)$**

std::vector & sorting con STL



std::vector

std::vector è una **sequenza** di oggetti dello stesso tipo (struttura dati parametrizzata sul tipo).

```
#include <iostream>
#include <vector>
using namespace std;
int main(){

    uint32_t n = 1000;

    vector<uint32_t> X(n); // creo un vettore lungo n di uint32_t inizializzati a 0
    vector<uint32_t> Y(n,5); // creo un vettore lungo n di uint32_t inizializzati a 5
    vector<uint32_t> Z; // creo un vettore vuoto uint32_t

    for(uint32_t i=0;i<n;++i)
        X[i] = i*i;

    for(auto x : X) // itera sugli elementi di X (nota la deduzione di tipo auto: da C++11)
        Z.push_back(x+7); // appendi in fondo a Z

}
```

std::vector

Possiamo parametrizzare std::vector su qualunque tipo, incluse struct:

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

struct persona{
    string nome;
    string cognome;
    int eta;
};

int main(){

    vector<persona> X; // creo un vettore di tipo persona
    X.push_back( {"Mario","Rossi",16} );// aggregate initialization
}
```

std::sort

Ordinare un vettore di interi in modo crescente (riuscite a trovarne la complessità sul manuale?)

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

int main(){

    vector<uint32_t> X;
    for(uint32_t i = 0;i<1000;++i)
        X.push_back(1000-i);

    vector<uint32_t> Y = X;

    sort(X.begin(),X.end()); //ordina in modo crescente l'intero vettore
    sort(Y.begin()+5,Y.end()-10); //ordina in modo crescente l'intero vettore esclusi i primi 5 elementi e gli ultimi 10

}
```

std::sort con comparazione arbitraria

Possiamo usare una funzione di ordinamento arbitraria? Sì!

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

```
using namespace std;
```

```
int main(){
    vector<int32_t> X;
    for(int32_t i = 0; i < 1000; ++i)
        X.push_back(500-i);
    auto comparator = [](const int32_t& lhs, const int32_t& rhs) { //comparatore per valore assoluto
        return abs(lhs) < abs(rhs);
    };
    sort(X.begin(), X.end(), comparator); //ordina usando il mio comparatore customizzato
}
```

Insieme



inserimento

- `std::vector` ci permette anche di inserire un elemento in mezzo al vettore (`std::vector::insert`)
- Ma a quale complessità? Cercatela sul manuale.

inserimento

- `std::vector` ci permette anche di inserire un elemento in mezzo al vettore (`std::vector::insert`)
- Ma a quale complessità? Cercatela sul manuale. **$O(n)$** !!
- Posso fare di meglio? Sì, usando strutture dati per insiemi (ordinati) e dizionari (non ordinati).

std::set

Insiemi di elementi ordinati con operazioni a **complessità logaritmica** ($O(\log n)$)

```
#include <iostream>
#include <set>
using namespace std;
int main(){

    auto X = set<int32_t>(); //mantiene gli elementi ordinati in modo crescente

    auto comparator = [](const int32_t& lhs, const int32_t& rhs) { //comparatore per valore assoluto
        return abs(lhs) < abs(rhs);
    };

    auto Y = set<int32_t,decltype(comparator)>(comparator); //ordina per valore assoluto crescente

    X.insert(-1000);
    X.insert(700); // in che ordine si trovano?
    Y.insert(-1000);
    Y.insert(700); // in che ordine si trovano?

}
```

std::set

Estrarre e rimuovere l'elemento più piccolo, **cercare** un elemento ($O(\log n)$)

```
Y.insert(-1000);  
Y.insert(-10);  
Y.insert(1200);  
Y.insert(188);
```

```
cout << "Elemento piu' piccolo: " << *Y.begin() << endl; //Y.begin() restituisce un iteratore (simile a puntatore) che va dereferenziato con *  
Y.erase(Y.begin()); //rimuove l'elemento più piccolo
```

```
if(Y.find(-1000) == Y.end())  
    cout << "Elemento non presente!";  
else  
    cout << "Elemento presente!";
```

std::multiset

- std::set non ammette duplicati (è un insieme!).
- Se inserite più volte lo stesso elemento in std::set, ne viene mantenuta solo una copia.
- std::multiset è simile a std::set, ma ammette duplicati.

Dizionari



`std::unordered_map`

- Si può migliorare ulteriormente la complessità $O(\log n)$ di un inserimento?

std::unordered_map

- Si può migliorare ulteriormente la complessità $O(\log n)$ di un inserimento?
- Sì, se abbandoniamo l'ordinamento!
- std::unordered_map è una **tabella hash**: inserimenti in tempo $O(1)$ (ammortizzato)

std::unordered_map

- Si può migliorare ulteriormente la complessità $O(\log n)$ di un inserimento?
- Sì, se abbandoniamo l'ordinamento!
- std::unordered_map è una **tabella hash**: inserimenti in tempo $O(1)$ (ammortizzato)
- Come suggerisce il nome, std::unordered_map non mantiene gli elementi in un ordine preciso: semplicemente li inserisce in un dizionario (crea un'associazione key-value).
- Ulteriore vantaggio: accesso tramite tipi arbitrari (non solo interi come in un vettore)

std::unordered_map

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

int main(){

    unordered_map<string, int> eta; // <key,value>

    eta["Mario"] = 30; //nuovo elemento: inserimento
    eta["Gianni"] = 25; //nuovo elemento: inserimento
    eta["Tom"] = 21;  //nuovo elemento: inserimento
    eta["Gianni"] = 19; //elemento esistente: sovrascritto

}
```

Code e stack



std::queue e std::stack

- Infine, altre strutture utili sono le code (FIFO) e gli stack (LIFO).
- Potete immaginarli come vettori in cui possiamo solo aggiungere un elemento in fondo (push) e:
 - Coda: rimuovere (pop) l'elemento più a sinistra (il più vecchio)
 - Stack: rimuovere (pop) l'elemento più a destra (il più nuovo)
- Il tutto in tempo $O(1)$!

std::queue e std::stack

- Infine, altre strutture utili sono le code (FIFO) e gli stack (LIFO).
- Potete immaginarli come vettori in cui possiamo solo aggiungere un elemento in fondo (push) e:
 - Coda: rimuovere (pop) l'elemento più a sinistra (il più vecchio)
 - Stack: rimuovere (pop) l'elemento più a destra (il più nuovo)
- Il tutto in tempo $O(1)$!

Più info nel manuale:

- <http://www.cplusplus.com/reference/queue/queue/>
- <http://www.cplusplus.com/reference/stack/stack/>

Esercizi olimpiadi



Esercizio 2

Possiamo finalmente risolvere dei semplici esercizi OII!

- Festa estiva(festaestiva) :[festa estiva](#)

Alcuni esercizi OII per allenarsi a casa:

- Soste in autostrada (autogrill): <https://training.olinfo.it/-/task/autogrill/statement>
- Tieni aggiornato il catalogo (catalogo): <https://training.olinfo.it/-/task/catalogo/statement>
- Musical fight (trap): [musical fight](#)
- Kill those bugs (blindpunch): [bugs](#)