

Quarta lezione

17 March 2022 13:57

Dichiarando una stringa senza inserire un parametro

- O inserendo solo string senza inizializzazione
- Inizializzando con un valore di default

Se ho già inizializzato una stringa posso anche andare a sovrascriverla da tastiera tramite il comando `cin>>stringa`

Lo stato interno della stringa non viene cancellato ma resta vuoto

In alcuni casi la cancellazione porta l'eliminazione della memoria come ad esempio nella struttura ad albero; infatti la cancellazione dell'albero prevede anche la cancellazione dei nodi e della radice (in caso di vettori, anche la deallocazione degli array).

ECCEZIONI E RELATIVE GESTIONI:

La struttura di una gestione elementare è simile a quella Java dunque si va ad utilizzare il try, nel quale posso sollevare un'eccezione con il throw, e il catch dove si vanno ad analizzare le seguenti eccezioni e se riaggiungo un altro throw posso andare a risollevare l'eccezione qual'ora fosse ingestibile.

Non tutte le funzioni possono sollevarle. **Se interrogo la classe stringa su quale dimensione abbia, non ha senso che l'operazione size sollevi l'eccezione, in questo caso posso andare a scrivere `type f(parameters) noexcept`.**

Non vado ad indicare quante classi sollevano l'eccezione ma solo se le sollevano o no.

ECCEZIONI STANDARD:

exception

logic_error
length_error
out_of_range

runtime_error

overflow_error
underflow_error

bad_alloc

PUNTATORI A FUNZIONE:

Possibili usi: passaggio di funzione come parametro di altre funzioni, funzioni callback

Definizione: `typedef type (*FunName) (lista parametri);` **METODO C**

`typedef std::function <type(lista parametri)> FunName;` METODO C++

Richiede `#include <functional>`

Un puntatore a funzione permette la chiamata di funzioni diversi con lo stesso prototipo conoscendo l'indirizzo.

ESEMPIO:

- `void sort (int*A, int n);` quando ho un ordinamento prefissato.
- `void sort(int*A, int n, compare cmp){....`
...`if(cmp(x,y)){...}else{...};`

SOLO IN PARTIZIONE CI SONO I CONFRONTI

o prefissato • Chiamate: `*FunName(...)` o `FunName(...)`.

! ...
• `bool lessThen (int x, int y) { return (x < y); }`
• `bool greaterThen (int x, int y) { return (x > y); }`
• `bool compare (int x, int y) { ... }`

• `Compare fun = [0]lessThen;`
• `sort(A, 10, [0]lessThen);` `sort(A, 10, fun);`
• `sort(A, 10, [0]greaterThen);`

CLASSE, OGGETTO E TEMPLATE

`class ClassName{`

[private]

...

protected;

...

public;

...

`};`

`public:`

... // attributi e metodi visibili a tutti.

... //

• `Constructor: ~ClassName();` // Destructore

• `Operator: E.g.: bool operator==(const ClassName &) const noexcept; // Comparison
ClassName& operator=(const ClassName &); // Copy assignment
ClassName& operator=(ClassName &&) noexcept; // Move assignment`

Constructors:
`ClassName();` // Default constructor
`ClassName(parameters);` // Specific constructors
`ClassName(const ClassName&);` // Copy constructor
`ClassName(ClassName&&) noexcept;` // Move constructor



Copia superficiale: solo a livello di superficie i due oggetti sono uguali

Copia profonda: vado a fare una copia completa, quindi vado a copiare anche il loro interno come i puntatori etc.

```
int *a= new int;
```

```
a= new int OPPURE a=b
```

UNO DIETRO L'ALTRO, NON SI DEVE FARE IN QUANTO L'INDIRIZZO VIENE PERSO (LEAK DI MEMORIA)!