Scelte implementative per il progetto C++

Federico Salvioni nnn aaa.bbb@campus.unimib.it

2025-01-07

I requisiti progettuali imprescindibili sono, sostanzialmente, i seguenti:

1. Non utilizzare costrutti avanzati dello standard C++11 e seguenti, ad eccezione di `nullptr`;
2. Non utilizzare una struttura dati a lista, quindi basata su nodi e puntatori;
3. Correttezza a discapito dell'efficienza.

In linea con la sintassi in genere utilizzata nel codice C++, soprattutto nella libreria standard, é stato preferito lo *sneak case* (`hello_world`) al *camel case* (`helloWorld`). Inoltre, come da programmazione ad oggetti, gli elementi atomici hanno un identificatore che inizia con la lettera minuscola, mentre gli oggetti container con la lettera maiuscola.

La struttura dati che é stata usata per implementare lo stack é sostanzialmente un array statico `Items` di dimensione fissata `maximum_size`, corredato di un intero `top_pos` che indica la posizione corrente della cima dello stack.

Il valore di `top_pos` viene inizializzato a -1, anziché 0. Questa potrebbe apparire una scelta singolare, ma Questo perché occorre poter avere a che fare con un array che non contiene alcun elemento. Inoltre, in questo modo, é possibile mappare la posizione dello stack sulla falsa riga di quella di un normale array, facendo partire la posizione da 0 anziché da 1.

`top_pos` parte da -1, mentre `maximum_size` parte da 0. Potrebbe essere quindi una scelta ragionevole dichiarare `maximum_size` come `unsigned`. Nonostante questo, entrambi sono dichiarati con lo stesso tipo `item_type` (`int`). Questo permette, nonostante tecnicamente ci sia un uso della memoria meno efficiente, di risparmiare molte annose conversioni di tipo, che fra i due sono fatte molto di frequente.

Il codice dispone di due eccezioni, `Maximum_size_reached` (requisito d'esame) e `Minimum_size_reached`. I due sono implementati come delle `struct` vuote, dato che non é necessario che contengano nulla.

Costruttori

Oltre ai due costruttori principali (default constructor e copy constructor) ed al costruttore che ha due iteratori per input (requisito d'esame), ne é stato aggiunto un quarto che inizializza uno stack vuoto di una certa dimensione. Tale costruttore é dichiarato `explicit` per fare in modo che il compilatore non lo intenda come un cast implicito. Le celle vuote sono inizializzate con il valore di valore di default del tipo templato, quale che sia.

Metodi

Tutti i metodi della classe `Stack` sono pubblici, dato che non vi sono particolari problemi se usati su una istanza di `Stack` al di fuori della classe stessa.

Dovendo implementare una struttura dati stack, é stato necessario dotarla delle seguenti operazioni:

- `push`, che aggiunge un elemento in cima allo stack;
- `pop`, che rimuove l'elemento sulla cima dello stack e lo ritorna;
- `peek`, che ritorna l'elemento in cima allo stack senza rimuoverlo;
- `stack_empty`, che ritorna se lo stack sia vuoto oppure no.

Oltre a queste, che sono metodi che ogni implementazione di uno stack deve predisporre, sono stati introdotti i seguenti metodi di supporto:

- `size`, che restituisce la dimensione massima dello stack;
- `head`, che restituisce la posizione della cima dello stack;
- `wipe`, che cancella il contenuto dello stack e lo ridimensiona a 0.

Si é cercato di evitare di introdurre ogni possibile metodo, preferendo invece dotare la classe del minimo numero possibile di metodi che fossero effettivamente utili. Ad esempio, per conoscere il numero di elementi al momento presenti nello stack `S`, anziché introdurre un metodo `S.current_capacity()` é sufficiente calcolare `S.head() + 1`.

Infine, i seguenti metodi sono stati introdotti perché requisiti d'esame:

- `load`, che carica lo stack a partire da una coppia di iteratori ad una sequenza;

- `clear`, che svuota lo stack del suo contenuto ma lascia intatta la sua dimensione;
- `filter_out`, che rimuove dallo stack tutti gli elementi che non rispettano un predicato.

Tutti i metodi hanno un assert che confronta `top_pos` con `maximum_size` per assicurarsi che il primo sia minore del secondo, dato che non potrá mai verificarsi una situazione dove questo non accade. Nei costruttori questo assert é assente perché irrilevante, dato che i valori vengono automaticamente inizializzati.

Funzioni globali

La classe `Stack` é stata dotata di una funzione globale `transform` (requisito d'esame) che applica una certa operazione ad ogni elemento di uno stack. Inoltre, é stato ridefinito l'operatore << per poter stampare a schermo il contenuto dello stack senza dover accedere ai suoi dati interni.

La stampa mediante << restituisce i valori all'interno di una coppia di parentesi quadre; se quel valore é oltre `top_pos`, le parentesi quadre non racchiudono nulla. Lo stack é restituito in orizzontale anziché in verticale per una semplice questione di leggibilitá.

Essendo funzioni globali e non metodi di classe, sia `operator<<` che `transform` accedono al contenuto dello stack mediante iteratori. Questo permette sia di evitare un *coupling* non necessario fra la classe `Stack` e tali funzioni, sia di avere maggiore flessibilitá, di modo che se la struttura interna della classe dovesse cambiare le funzioni non debbano venire aggiornate di conseguenza.