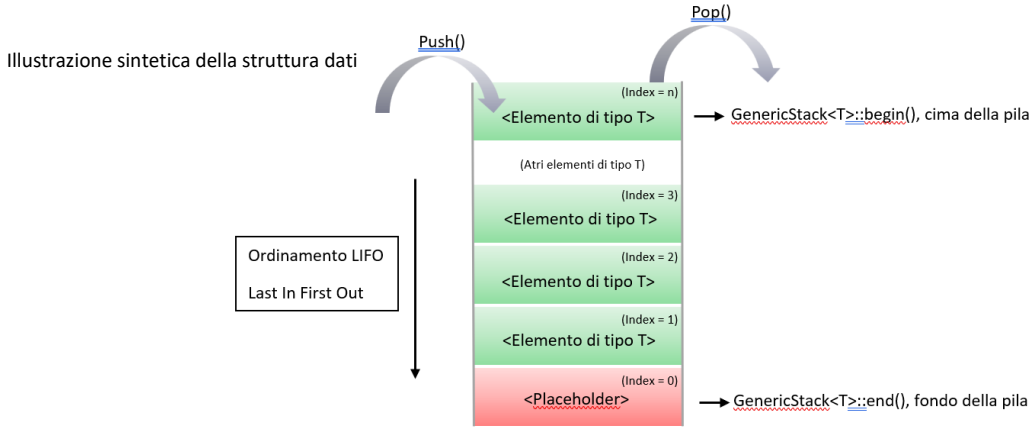
Pietro Bressan 852260 p.bressan@campus.unimib.it

Relazione GenericStack

Nel seguente documento si presentano le scelte implementative seguite per la realizzazione della classe GenericStack. In particolare si tratta di una classe template che implementa una pila, il cui contenuto quindi viene specificato in fase di utilizzo della stessa.

Gestione della struttura dati



La classe intende rappresentare una pila di dati generici, la cui dimensione è nota in fase di creazione.

Il tipo di dato della dimensione è volutamente 'mascherato' dal tipo custom *size_type*, con la possibilità, in futuro, di modificarlo senza impattare un ipotetico utilizzatore finale della classe.

Sempre per via della dimensione che viene specificata in fase di creazione, a livello programmativo si è optato per rappresentare la struttura dati mediante un array di dati generici *_stack*. Rispetto ad altre possibili soluzioni questa si presenta come la più semplice e pratica: anzitutto abbiamo già a disposizione un oggetto come l'array che gestisce l'ordinamento in memoria dei dati che contiene, così come l'accesso ad essi, senza necessità di creare classi 'container' (come può essere un nodo in una lista); inoltre si ha la comodità di occupare immediatamente lo spazio richiesto per tutta la pila, anche qualora sia popolata in momenti differenti dalla sua istanziazione. Questa caratteristica rispecchia ancora una volta la consegna, ed assicura all'utente finale di avere in mano un oggetto nel quale può inserire i dati di cui ha bisogno senza preoccuparsi della memoria.

Il costruttore di default è quindi sostituito da un costruttore che prende come parametro di ingresso la dimensione della pila.

Per conoscere il numero di elementi presenti nella pila e il numero di elementi totali che questa può contenere sono state utilizzate due variabili, rispettivamente *_current_size* e *_stack_size*.

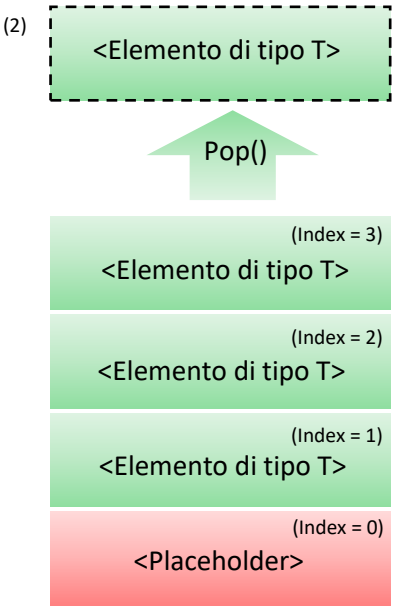
Al fine di tenere traccia del fondo della struttura dati, e distinguerlo dall'ultimo elemento nella pila, è stato scelto di riservare l'ultima posizione della pila come valore di placeholder cui compito è comunicare alla classe la fine degli elementi prelevabili. Ciò si traduce nella variabile *_stack_size* incrementata di uno rispetto alla dimensione fornita dall'utente, e nella prima cella dell'array *_stack* che non verrà mai davvero occupata da nessun elemento. Questa scelta implementativa può non essere efficiente qualora il tipo di dato su cui si specializza la classe richieda molto spazio, ciononostante per la maggior parte dei casi questa metodologia non dovrebbe rappresentare un grave spreco di memoria e rimane tutto sommato di facile realizzazione. Il meccanismo è infine nascosto all'utilizzatore al quale come dimensione della pila verrà correttamente ritornato il valore da lui scelto.

L'utente che chiamerà la pila non avrà modo di accorgersi dell'array sottostante, grazie ai metodi che questa espone e che obbligano gli inserimenti e le eliminazioni ad avvenire per ordinamento LIFO.

Per inserire un elemento si utilizza il metodo *push()* nel quale si controlla che la pila non sia piena e in tal caso si procede alla sua aggiunta nella struttura dati. Qualora invece la pila non avesse più spazio utilizzabile verrebbe generato un errore.

Per togliere un elemento dalla cima della pila e ritornarlo è possibile ricorrere al metodo *pop()*, dove anche qui viene fatto un controllo analogo nel caso si cerchi di eliminare elementi ma la pila è vuota.

Tutti gli elementi nella struttura dati sono definiti come const. Essendo la classe pensata come un "contenitore" di dati si è preferito non lasciare la possibilità di modificarli, ma solo di aggiungerli o rimuoverli.



Nel caso di uno stack di 4 elementi

(1) *_stack_size* = 5
_current_size = 4
begin() = array[4]
end() = placeholder

(2) *_stack_size* = 5
_current_size = 3
begin() = array[3]
end() = placeholder

Per scorrere gli elementi è stata definita all'interno della pila una classe *const_iterator* e anche la sua implementazione vuole seguire il più possibile l'ordine della struttura dati. Per questo motivo l'iteratore di inizio parte dalla cima della pila, mentre quello di fine punta all'elemento placeholder. Per muoversi tramite l'iteratore è necessario chiamare l'operatore di decremento che, dall'ultimo elemento dell'array *_stack*, scorre "dall'alto al basso" la pila. Onde evitare operazioni non consentite e soprattutto al fine di notificarle all'utente qualora queste avvengono, anche il *const_iterator* può generare un errore qualora si provi a decrementare l'iteratore di end o si provi a dereferenziarlo.

È presente un metodo *flush()* che svuota logicamente la pila ponendo *_current_size* = 0.

La classe è poi dotata di un metodo template *checkIf()* che prende come parametri di ingresso un elemento dello stesso tipo su cui è specializzata la pila e un altro dato, generico, di tipo P. Questo dato sarà in realtà un predicato utilizzato per valutare l'elemento passato in input.

È stato ridefinito a livello globale l'operatore di stream al fine di consentire la gestione di un GenericStack e così da permettere la sua visualizzazione a terminale. Nei commenti del codice viene comunque avvisato l'utente qualora lo utilizzi, ricordandogli di assicurarsi che sia definito l'operatore anche per il tipo di dato che contiene la pila.