Bazzocchi Tommaso 869095 t.bazzocchi@campus.unimib.it

RELAZIONE PROGETTO PARTE C++:

Stack - Stack<T>

Classe template utilizzata per rappresentare uno stack con all'interno dati di un qualunque tipo.

DATI MEMBRO:

T* stack;

int size;

int indice;

Per rappresentare lo stack ho deciso di utilizzare un array dinamico. A inserire gli elementi parto dalla posizione 0 dell'array, quindi questa sarà la coda dello stack e man mano che faccio una push vado a riempire l'array, quindi parto dalla posizione 0 e ogni volta che devo inserire un elemento incremento la posizione. Per rappresentare dove inserire il prossimo elemento utilizzo il dato membro indice.

Indice-1 sarà la testa dello stack mentre 0 sarà la coda. Size indica la dimensione dell'array.

Es: se io voglio inserire in un array di 5 elementi 1 2 3 4 5 in questo ordine dove 1 è la coda e 5 la testa,

l'array sarà riempito cosi:

A[0]= 1 ; A[1]= 2 ; A[2]= 3; A[3]= 4; A[4]= 5.

L'elemento in posizione 4 sarà la testa, mentre l'elemento in posizione 0 sarà la coda.

E quando farò la pop quindi partirò dall'elemento in posizione 4.

COSTRUTTORE DI DEFAULT:

Il costruttore di default crea uno stack vuoto, settando la dimensione dello stack a 0, e l'indice a 0.

COSTRUTTORE SECONDARIO (1):

Questo costruttore secondario prende come input la dimensione che vogliamo che abbia lo stack. Questo costruttore serve per allocare dinamicamente l'array la quale dimensione è quella passata come parametro. E l'indice è settato a 0, cioè la posizione in cui dovrà essere inserito il prossimo elemento sarà la 0.

L'allocazione dell'array è inserita all'interno di un try/catch, perché la new potrebbe generare un'eccezione. In questo modo nel caso in cui l'allocazione non andasse a buon fine lo stack verrà riportato in uno stato coerente utilizzando la funzione clear().

COSTRUTTORE SECONDARIO (2):

Questo costruttore secondario ha come parametri una coppia di iteratori, che puntano uno all'inizio di una sequenza di elementi e uno alla fine della sequenza, e la dimensione che vogliamo che abbia lo stack. Questo costruttore è usato per riempire lo stack con i valori della sequenza a cui puntano i due iteratori.

Questo costruttore riempie lo stack dalla dimensione 0 dell'array con i valori presenti nella sequenza a partire dal valore a cui punta l'iteratore di partenza passato come parametro.

Nel caso in cui la dimensione della sequenza sia piu' grande dell'array che sto andando a istanziare mi fermo quando arrivo alla posizione size-1.

Anche qua utilizzo try/catch perché la new potrebbe generare un'eccezione. In questo modo nel caso in cui l'allocazione non andasse a buon fine lo stack verrà riportato in uno stato coerente utilizzando la funzione clear().

COPY CONSTRUCTOR:

Il copy constructor prende in input un altro oggetto Stack e crea un nuovo oggetto identico ma distinto impostando la dimensione dell'array con quella dell'oggetto passato.

Quindi viene allocato un array con dimensione identica a quella dello Stack passato.

E indice viene settato a 0.

Anche qua utilizzo try/catch perché la new potrebbe generare un'eccezione. In questo modo nel caso in cui l'allocazione non andasse a buon fine lo stack verrà riportato in uno stato coerente utilizzando la funzione clear().

OPERATORE DI ASSEGNAMENTO:

L'operatore di assegnamento prende come parametro di input un altro oggetto Stack. Dopo aver verificato che l'oggetto a cui si sta assegnando non è lo stesso che si sta tentando di assegnare, creo una copia Stack temporanea dell'elemento passato utilizzando il copy constructor, dopodiché viene chiamata la funzione swap con parametro la variabile temporanea di tipo Stack appena creata. Quest'ultimo si occupa dello scambio dei dati membro tra i 2 oggetti.

Infine, restituisce la dereferenziazione del puntatore this all'oggetto.

PUSH:

Questo metodo prende come argomento un elemento dello stesso tipo di stack. La push aggiunge un elemento in testa allo stack, quindi l'elemento viene inserito nella posizione indicata dal dato membro indice.

L'inserimento dell'elemento lo metto all'interno di un blocco try/catch perché nel caso in cui lo stack sia pieno viene generata un eccezione che stampa a schermo: lo stack e' pieno, quindi non e' possibile fare la push.

POP:

Questo metodo restituisce ed elimina dallo stack l'elemento in testa. Elimina dallo stack l'elemento in posizione indice – 1, quindi l'ultimo elemento inserito.

Le linee di codice che riguardano l'eliminazione dell'elemento in testa le metto all'interno di un blocco try/ catch, perché potrebbe darsi che lo stack è vuoto e quindi in quel caso genero un eccezione che stampa a schermo: lo stack e' vuoto non ci sono elementi, quindi non e' possibile fare la pop.

GETSIZE:

E' un metodo getter, che ritorna la dimensione dello stack.

GETPOSIZIONEULTIMOELEMENTO:

E' un metodo getter, che ritorna la posizione dell'elemento in testa allo stack, quindi indice-1.

GETINDICE:

E' un metodo getter, ritorna la posizione in cui sarà inserito il prossimo elemento.

SVUOTA STACK:

Questo metodo viene usato per svuotare lo stack, io ho deciso di svuotare lo stack logicamente quindi per svuotarlo setto l'indice a 0,e questo vuol dire che il prossimo elemento da inserire sarà messo nella posizione 0.

FILLSTACK:

E' un metodo templato. Questo metodo viene usato per riempire lo stack con una sequenza di elementi puntata da una coppia di iteratori. Prende come argomento una coppia di iteratori che punta uno all'inizio della sequenza e uno alla fine. Riempie lo stack a partire dalla posizione 0 con gli elementi della sequenza a partire da quello puntato dall'iteratore che punta all'inizio della sequenza. Ci si ferma quando o si è arrivati alla fine della sequenza o lo stack è pieno.

Nel caso in qui lo stack non sia vuoto, lo si svuota attraverso il metodo svuotaStack e poi lo si riempie come detto in precedenza.

STAMPASTACK:

Questo metodo viene utilizzato per stampare lo stack a partire dalla testa. Per fare ciò viene utilizzato un iteratore di sola lettura che all'inizio viene fatto puntare attraverso il metodo end() degli iteratori costanti alla testa dello stack. Decremento l'iteratore finchè non punta alla coda dello stack e per controllare questo utilizzo il metodo begin().

Nel caso in cui lo stack sia vuoto, quindi controllo che l'indice sia uguale a 0, stampo a schermo: "lo stack è vuoto, al momento non ci sono elementi".

Bazzocchi Tommaso 869095 t.bazzocchi@campus.unimib.it

CHECKIF:

Funzione templata. Prende in input un elemento dello stesso tipo di stack e un predicato generico. Ritorna true/false in base a se l'elemento soddisfa il predicato o no.

ITERATORI:

Come iteratori ho usato un iterator e un const_iterator di tipo forward iterator. Ho fatto questa scelta perché questo tipo di iteratori offrono una maggiore flessibilità rispetto agli altri tipi di iteratori, in quanto possono essere utilizzati sia per iterare in avanti sia per iterare indietro. Nel metodo stampaStack() a noi serve iterare indietro e quindi ho scelto loro.

Ho dotato la classe templata Stack di un iterator e un const iterator che possono essere utilizzati per puntare ai vari elementi dello stack. Il const_iterator viene utilizzato nel metodo stampaStack().

Iterator e const_iterator sono dotati di due metodi begin(), ed end():

- Begin(): ritorna un iteratore che punta alla prima cella dello stack.
- End(): ritorna un iteratore che punta alla posizione indicata dall'indice, quindi dove sarà inserito il prossimo elemento.

L'iteratore restituito dalla funzione end() punta a una cella che non fa parte della sequenza, perché punta alla posizione dove sarà inserito eventualmente un prossimo elemento, però questo non è un problema perché l'iteratore restituito da questa funzione verrà utilizzato solo per capire quando la fine della sequenza è stata raggiunta.

Bazzocchi Tommaso 869095 t.bazzocchi@campus.unimib.it