Relazione progetto circular buffer

Nome: Fabio
Cognome: Villa
Matricola: 829583
Mail campus: f.villa71@campus.unimib.it

Introduzione

Per implementare la classe circular buffer (coda FIFO), ho deciso di definirla con due parametri template: T per il tipo generico dei valori e Q per il funtore di uguaglianza che può essere deciso dall'utente (necessario per find).
Questo progetto implementa i metodi fondamentali, quali costruttori, copy constructor e distruttori e metodi per aggiungere e cerca un elemento all'interno del buffer.

Element

Il buffer è costituito da elementi, implementati in una struct privata nella classe cbufferlist formata come:

- Il valore dell'elemento
- Il puntatore all'elemento successivo

Nel costruttore di default, i dati sono inizializzati tramite initialization list, per migliorare l'efficienza dell inizializzazione. Un elemento vuoto viene inizializzato con il costruttore di default, mentre il puntatore viene inizializzato a nullptr.

Nel costruttore secondario, anch'esso sono inizializzati con initialization list. Il valore di T viene passato al costruttore e il puntatore all'elemento successivo della coda è inizializzato a nullptr.

Nel costruttore secondario, anch'esso sono inizializzati con initialization list. Il valore di T viene passato al costruttore e il puntatore all'elemento successivo della coda è inizializzato al valore n passato come secondo parametro al costruttore.

Nel distruttore, chiamato automaticamente, corrispondente al valore distruttore del tipo T. Al puntatore all'elemento successivo della coda viene assegnato il valore di default nullptr.

Gli altri metodi fondamentali (Copy Constructor e operatore di assegnamento) coincidono con quelli di default è inutile specificarli.

Dati membro

I dati membro della coda sono un puntatore alla testa (necessario per la rimozione), un puntatore alla coda (necessario per inserimento), la dimensione e il funtore per il confronto di uguaglianza (per find) e un quanti elementi ci sono all'interno.

Implementazione cbufferlist

Il cbufferlist è una classe templata che richiede un valore (T) e un operatore di confronto (Q) e quattro tipi costruttori:

- Di default, che setta la testa e la coda a nullptr e la grandezza a 10 costruito con initialization list
- Un costruttore con la grandezza scelta dall'utente costruito con initialization list
- Di copia, che, dato un altro buffer come parametro, copia, se possibile, gli elementi del buffer copiandone anche la struttura.
- Un secondario templato, che permette di riempire il buffer dando un sequenza di dati

Inoltre ha un distruttore che elimina tutti gli elementi creati e allocati in memoria.

Metodi di cbufferlist

Il metodo **size**: ritorna il valore della lunghezza del buffer.

Il metodo **elementil**: ritorna quanti elementi ha il buffer.

Il metodo **get_head**: ritorna la testa senza estrarla, non necessario, ma utile per testare la classe.

Il metodo **get_tail**: ritorna la coda senza estrarla, non necessario, ma utile per testare la classe.

Il metodo **find**: ci dice se un dato valore esiste nel buffer, se esiste ritorna true altrimento false. Usa l'operatore di uguaglianza custom creato dall'utente e itera tutto il buffer per vedere se esiste il valore passato.

Tutti questi metodi sono sicuri, perché non cambiano lo stato del buffer, quindi possono essere usati anche su un buffer costante.

Il metodo **insert**: che inserisce un elemento nel buffer con un determinato valore T, ma se il buffer è pieno sovrascrive il primo elemento. Per implementare la insert vedo se la gli elementi all'interno sono minori della size se lo sono allora vado a guardare se la testa punta a nullptr se è vero aggiungo l'elemento alla testa (al termine coda e testa puntano allo stesso oggetto), se è falso la coda avanza e viene inserito l'oggetto nella coda a questo punto aumento gli elementi nel buffer.
Se gli elementi sono maggiori invece prendo la testa e la faccio puntare al secondo elemento, distruggo la vecchia testa e accoda il nuovo elemento, quindi la testa ora punterà

al vecchio secondo elemento e la coda al nuovo elemento inserito, qui gli elementi all'interno non aumentano, ma rimangono costanti.

Il metodo **extract**: estrarre e ritorna l'elemento più vecchio della coda. Implementata in maniera da andare a richiedere il valore della testa, salvarlo in una variabile, far puntare la testa all'elemento successivo, distruggere il puntatore alla vecchia testa e ritornare il valore. Se il buffer è vuoto solleva un'eccezione apposita.

L'**operatore []** ha la capacità sia tornare una valore all'indice del buffer che modificarlo. Se l'indice è maggiore della lunghezza viene sollevata un'eccezione

I metodi **clear e clear_helper (modificare):** queste due funzioni private vengono utilizzate sia dal distruttore, sia in alcuni casi di eccezione di allocazione di memoria.

clear rimuove tutti gli elementi del buffer. La rimozione viene effettuata ricorsivamente, chiamando la funzione helper privata ricorsiva clear_helper(element *e), a partire dall'elemento in testa alla coda. Al puntatore alla coda viene assegnato il valore di default nullptr (poiché, al termine dell'esecuzione della funzione helper ricorsiva, punta ad una locazione di memoria ora non più valida).

clear_helper rimuove ricorsivamente gli elementi del buffer a partire da un elemento specificato attraverso il puntatore ad element passato come parametro (compreso). Ogni volta che la funzione viene chiamata ricorsivamente, vengono svolte le seguenti operazioni: la funzione guarda se il puntatore punta a nullptr se non è così procede con il prossimo elemento fino a quando non punta nullptr, a quel punto elimina tutti gli elementi partendo dalla fine fino alla testa; infine, il numero di elementi inseriti nella cosa viene decrementato di un'unità.

Iteratori

Per gli iteratori, ho deciso di implementare sia quelli in lettura e scrittura, sia quelli costanti (dato che, su buffer const, gli iteratori normali non possono essere istanziati).

Gli iteratori sono di tipo forward, in quanto seguono la logica FIFO di ritornare i valori in un certo ordine (quindi non possono né accedere randomicamente ai valori, né procedere bidirezionalmente).

Se gli iteratori sforano le locazioni di memoria della coda, viene lanciata un'eccezione apposita, che impedisce di accedere a delle locazioni non valide e sconosciute.

Operatore di stream

Operatore<< che sfrutta gli iteratori per stampare la struttura, e nel caso, la sottostruttura del buffer. Non necessario, ma implementato per test.

main

File sorgente contenente la funzione main, vari tipi custom, con relativi operatori di uguaglianza. Nel main si va a testare la classe cbufferlist con vari tipi: interi, stringhe, struct di complessi e un cbufferlist di interi.