PROGETTO C++
FREBBRAIO 2018

Alessio Porta 807457
a.porta14@campus.unimib.it

INTRODUZIONE

Ho deciso di rappresentare il buffer circolare come un array.
La classe è un template dove viene definito un tipo T che rappresenta il tipo dei dati contenuti dell'array.

IMPLEMENTAZIONE

Come detto sopra il buffer circolare è rappresentato con un array, all'interno della mia classe viene salvato e usato tramite un puntatore ad un array di tipo T(che sarebbe il tipo templato dei dati, come detto sopra) .

Oltre al puntatore all'array ho deciso si tenere anche la dimensione di quest'ultimo, per l'inizializzazione e alcuni metodi, tramite la variabile _size e il numero di elementi occupati tramite la variabile _end.

Il buffer circolare rappresentato dell'array ha sempre come elemento più vecchio quello in prima posizione(cioè posizione zero), e l'ultimo in posizione _end – 1.
Ho scelto questo metodo per rendere più facile la gestione dei dati al suo interno e l'utilizzo degli iteratori.
L'array viene istanziato con una certa dimensione data tramite dei costruttori secondari(o zero con il costruttore di default) , non ci sono metodi per cambiare la dimensione dell'array, li ho ritenuti superflui perché il buffer di solito ha una dimensione costante che non varia.
Nel caso si volesse cambiare la dimensione è possibile costruire un nuovo oggetto della mia classe buffer circolare e passargli i dati tramite il costruttore con iteratori.

Per la gestione dei dati sul buffer sono stati implementati i metodi insert e remove come richiesto.

Il metodo insert viene eseguito solo se la _size è maggiore di 0 e si divide in due casi: quando il buffer non è pieno aggiungo semplicemente l'elemento alla posizione di _end e incremento _end di uno, se è pieno il valore dovrebbe sostituire quello più vecchio, perché essendo circolare l'ultima posizione corrisponde alla prima, perciò faccio uno shift a sinistra e inserisco il nuovo elemento nell'ultima posizione dell'array.
Faccio uno shift a sinistra di una posizione perché così facendo il secondo elemento finisce in prima posizione e diventa quello più vecchio.

Il metodo remove invece viene eseguito solo nel caso che non sia vuoto l'array, ed esegue lo shift a sinistra di una posizione e decrementa _end di uno.
Cosi facendo rimuovo l'elemento in testa, cioè sposto tutti gli elementi di una posizione, non è una rimozione fisica della cella dalla memoria ma solo una sostituzione dei valori in essi contenuti.

Per l'accesso a una cella del buffer circolare ho usato l'operatore[] che ha come parametro l'indice della posizione a cui accedere.
Posso accedere solo a valori tra zero e _end(questo escluso) , anche se è circolare non mi è sembrato sensato poter "girare" su di esso con indici maggiori, cioè se ho un buffer grande cinque con tre elementi inseriti, non posso accedere al valore in posizione due tramite l'operatore[] con indice sette, perché pur essendo circolare non ho circolarità nella lettura dei dati ma solo nella gestione tramite insert e remove.

ITERATORI

Ho deciso di usare gli iteratori random access, perché in buffer posso accedere liberamente a qualsiasi elemento oltre all'accesso iterativo degli iterati direzionali.
Come variabile usata nell' iteratore per accedere ai dati del buffer circolare ho usato un puntatore all' array di tipo T.
Tramite questo puntatore posso accedere agli elementi dell' array.
Il metodo begin restituisce un nuovo puntatore al primo elemento dell' array che nel buffer circolare corrisponde a quello più vecchio.
Il metodo end invece restituisce un puntatore alla posizione di _end, che corrisponderebbe alla dimensione attuale occupata dal buffer.
Nei metodi dell' iteratore che spostano il puntatore non vengono eseguiti controllo che sfori dall'array perché deve essere gestito dall'utente.

FUNZIONI ESTERNE

Ho anche implementato l'operatore<< per scrivere i dati del buffer su uno stream di output, tramite accesso con gli iteratori begin e end, nel caso sia vuoto il buffer viene restituito un messaggio.

La funzione evalute_if è stata definita anch'essa template con il tipo degli elementi del buffer e il tipo P del predicato unario.
Questa funzione è stata implementata tramite dei const_iterator.

MAIN

Nel main ho testato alcuni casi di utilizzo della classe.
Oltre ai test dei metodi e dei costruttore della classe con interi ho voluto anche testare gli iteratori e la funzione evaluate_if.
Quest'ultima prende un oggetto di tipo cbuffer e un predicato unario come parametri.
I predicati unari sono stati implementati come funtori con un solo parametro.
I due funtori che ho creato per testare questa funzione sono greater_zero che dato come parametro un intero restituisce true se è maggiore di zero e less_zero che fa l'opposto.

Ho anche testato la classe con il tipo voce, che è una classe creata a lezione e usata in altri contesti, per questo motivo nella cartella ci sono anche file di questa classe.