RELAZIONE PROGETTO PROGRAMMAZIONE E AMMINISTRAZIONE DI SISTEMA C++

FEBBRAIO 2018 A.A 2017/2018

-----------------------------------------------------------------------------------------------------------------------------------------------

**Nome: Davide**

**Cognome: Ditolve**

**Mail: d.ditolve@campus.unimib.it**

**Matricola: 806953**

----------------------------------------------------------------------------------------------------------------------------------------------

-----------------------------------------------------< **INTRODUZIONE** >----------------------------------------------------------

Il progetto richiede di implementare una classe

---------------------------------------------------< **RAPPRESENTAZIONE DEI DATI** >-------------------------------------------

L’array rappresenta il circular buffer, l’utilizzo di puntatori a testa e coda consente di implementare il circular buffer correttamente in quanto chiedendo la posizione successiva alla coda per esempio dovrebbe dare out of bounds invece per definizione di circolare questa operazione riporta il pointer di posizionamento alla testa

-----------------------------------------------------------------------------------------------------------------------------------------------

--------------------------<**CLASSE CIRCULAR BUFFE E METODI FONDAMENTALI**>------------------------------------

Il circular buffer contiene come dati:

Cbuffer: l’array di elementi

Dimensione: quanti elementi puo’ avere il buffer

Testa e coda: puntatori rispettivamente al primo e all’ultimo elemento dell’array

Occupati: tiene traccia di quanti elementi ci sono e che non sono da cancellare

Logger: variabile che contiene il logger che consente di stampare a video eventuali problemi o info.

Metodi base implementati:

Costruttore di default che setta tutto a 0

Costruttore parametrico che prende in input la grandezza dell’array da istanziare

Copy costructor

Per la corretta implementazione dei controlli all’interno dei metodi ho deciso di creare dei metodi ausiliari che consentono di ottenere lo stato corrente del buffer in particolare questi metodi sono:

get dimensione: restituisce la grandezza del buffer

get occupati: restituisce quanti elementi ci sono all’interno del buffer

get spaziolibero: restituisce la dimensione – gli occupati

get testa: restituisce il puntatore alla testa se il pointer è null viene lanciata un’eccezione

get coda: restituisce il puntatore alla coda se il pointer è null viene lanciata un’eccezione

distruttore di default: effettua una chiamata a dispose

Dispose: cancella e setta a 0 tutte le info

----------------------------------------------------< **METODI AVANZATI** >-------------------------------------------------------

INSERT

Il metodo consente di inserire un elemento all’interno del buffer con più casi:

se il buffer ha dimensione 0 loggo l’errore e ritorno il controllo al chiamante

se il buffer ha spazio libero: se non ci sono occupati scrivo in posizione 0 altrimenti alla prima posizione libera

Se il buffer non ha spazio libero shifto tutto l’array indietro di una posizione e inserisco l’elemento in ultima posizione

DELETE HEAD

Controllo che l’operazione sia completabile e in tal caso shifto indeitro di una posizione tutto l’array così la posizione 0 viene sovrascritta

SHIFT AVANTI

Assegna all’elemento i-1 l’elemento i

SWAP

Scambia i valori di this con other

--------------------------------------------------< **OPERATORI** >--------------------------------------------------------------------

Sono stati implementati gli operatori:

[] per l’accesso indexato al buffer sia normale sia costante

= per l’assegnamento di variabili di tipo circularbuffer

----------------------------------------------------< ITERATORI >-------------------------------------------------------------------

Sono stati implementati due tipi di iteratori quello const e quello normale, entrambi hanno il costruttore di default, il costruttore con parametro value e il costruttore con puntamento, coda e testa che consente di salvare informazioni sullo stato necessario per eseguire ulteriori controlli nell’implementazione degli operatori

Sono stati implementati i seguenti operatori:

++: pre e post incremento per spostarsi in avanti all’interno del buffer, se il posizionamento raggiunge la coda il successivo ++ riporta il puntamento alla testa

-- :pre e post decremento per spostarsi indietro all’interno del buffer, se il posizionamento raggiunge la testa il successivo - - riporta il puntamento alla coda

== : operatore per il controllo di uguaglianza

!= : operatore per il controllo di disuguaglianza

\* : operatore per la dereferenziazione

[] : operatore per l’accesso indexato

= : operatore di assegnamento

----------------------------------------------------< **ECCEZIONI** >---------------------------------------------------------------------

Ho scelto di utilizzare le eccezioni contenute nella libreria STD senza definirne altre customizzate, e sono utilizzare per la corretta del gestione del flusso del programma e evitare che in caso di input sbagliati il programma finisca in uno stato non consistente o addirittura termini l’esecuzione.

-----------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------< **CLASSE LOGGER** >----------------------------------------------------------

Il logger risulta fondamentale in qualsiasi linguaggio di programmazione, l’implementazione molto base di questa classe consente di loggare sul terminale messaggi con 5 livelli di errore.

Un logger viene istanziato con un livello che va da INFO a ERROR, se viene richiesta la loggatura con un livello inferiore a quello desiderato non verrà scritto nulla sul terminale