RELAZIONE PROGETTO PROGRAMMAZIONE E AMMINISTRAZIONE DI SISTEMA C++

APRILE 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 sortedarray la quale deve consentire all'utente di inserire dati di tipo T in qualsiasi ordine lui voglia ma che li tenga ordinati (tramite un funtore) logicamente senza alterare l'array

---

< RAPPRESENTAZIONE DEI DATI >

L'array di tipo T contiene tutti i dati di tipo T inserite nell'ordine in cui l'utente ne richiede l'inserimento. L'array di interi invece tiene traccia delle posizioni nell'array disordinato affinchè la lettura in sequenza delle posizioni riproduca l'array disordinato in maniera ordinata

---

<CLASSE CIRCULAR BUFFE E METODI FONDAMENTALI>

Il sortedarray contiene come dati:

Unsorted_data_: Array di dati non ordinati

Sorted_data_ : Array di posizioni per tenere logicamente ordinato l'array Unsorted_data_

Dimensione_array_: quanti elementi puo' avere il buffer

Head e tail: puntatori rispettivamente al primo e all'ultimo elemento dell'array ordinato

UHead e utail: puntatori rispettivamente al primo e all'ultimo elemento dell'array non ordinato

Last_inserted_ : tiene traccia di quanti elementi sono stati inseriti

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 degli array in particolare questi metodi sono:

get dimension: restituisce la grandezza del array

get lastInserted: restituisce quanti elementi ci sono all'interno dell'array

get freespace: restituisce la dimensione – gli occupati

get head: restituisce il puntatore alla testa unsorted data

get tail: restituisce il puntatore alla coda di unsorted data

get uhead: restituisce il puntatore alla testa sorted data

get utail: restituisce il puntatore alla coda di sorted data

distruttore di default: effettua una chiamata a dispose

Dispose: cancella e setta a 0 tutte le info

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

INSERT DATA

Prende in input l'elemento da inserire, posiziona in coda all'array unsortedData l'elemento e poi utilizzando l'ordinamento definito dal funtore per posizionare correttamente la posizione all'interno dell'array di interi sortedData

EMPTY DATA

Consente di svuotare gli array e riportare lo stato dell'oggetto a quello successivo alla creazione. I dati vengono cancellati settando a 0 i relativi valori. Vengono ripristinati gli stati delle varie variabili.

SWAP

Scambia i valori di this con other

SHIFT ITEM

Consente di shiftare avanti a partire da una posizione tutti gli elementi dell'array di interi, rende così possibile l'inserimento della posizione dell'elemento appena inserito in posizione corretta tramite insert data.

SORTEDPRINT

Consente di stampare i dati di tipo T in maniera ordinata

UNSORTEDPRINT

Consente di stampare i dati di tipo T nell'ordine di inserimento

---

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

Sono stati implementati gli operatori:

[] per l'accesso indexato ai dati di tipo T in maniera ordinata sia normale sia costante

() per l'accesso indexato ai dati di tipo T in maniera non ordinata sia normale sia costante

= per l'assegnamento di variabili di tipo sortedarray

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

Sono stati implementati due tipi di iteratori quello const e quello unsorted const, entrambi hanno il costruttore di default. L'iteratore ordinato ha un costruttore che prende in input un puntatore ad interi posizionamento e un puntatore a dati di tipo T data, posizionamento è necessario altrimenti non è possibile ordinare i dati.

L'iteratore non ordinato ha un costruttore che prende in input solamente data in quanto non è necessario ordinare logicamente i dati

Entrambi gli iteratori sono stati implementati come random access iterator per fornire quante più funzionalità possibili

Entrambi gli iteratori sono di tipo const per impedire la scrittura e renderli quindi di sola lettura come richiesto dalle specifiche

---

----------< **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

---

----------< **TEST E CONSIDERAZIONI** >----------

La classe main contiene diversi funtori di ordinamento e una struct complex per effettuare test su strutture dati custom.

Ho cercato di effettuare test di ogni tipo, raggruppandoli in vari metodi. I primi test sono per i metodi di base, vengono testati gli inserimenti leciti e illeciti, viene testata la stampa ordinata e disordinata.

Come da specifica viene testato il metodo globale find_count sia su dati semplici (Int) sia su strutture dati complesse (Complex)

Gran parte dei test sono stati dedicati agli iteratori, in particolare i metodi test_iteratori2 e 3 creano un sortedArray e ne testano il comportamento con vari operatori il 2 con const_iterator e il 3 con u_const_iterator

I test più significativi per gli iteratori sono nel metodo iteratorTest, vengono creati dei sortedarray con il costruttore templato IterT, su questi testbuffer vengono creati vari iterator e vengono testati attraverso l'utilizzo di metodi nella libreria algorithms contenuta in std.

Il progetto si è rivelato molto impegnativo, soprattutto nella fase di test dato che le funzionalità implementate sono davvero tante e testarle tutte in maniera approfondita risulta complesso.