Relazione Progetto C++

Settembre 2020

Nome: Matthew
Cognome: Barbier
Matricola: 830111
E-mail: m.barbier@campus.unimib.it

========== || Introduzione || ==========

Il progetto consiste nell'implementazione di un buffer circolare di elementi generici e omogenei. Questo buffer sarà di dimensione fissa, distinguendosi da un normale buffer per la sua contiguità tra l'ultima locazione di memoria del buffer e la prima.

========== || Implementazione || ==========

Per implementare tale buffer ho deciso di utilizzare un array, permettendo così un accesso alle singole celle in un tempo relativamente ottimale *O(1)* sia in lettura che in scrittura.
Oltre alla dichiarazione del buffer come puntatore a un tipo generico, vengono dichiarate altre tre variabili di tipo *unsigned int*, che tramite l'istruzione *typedef* ho rinominato in *size_t*.
Tali variabili sono dunque:

- ➔ *_capacity* : per indicare la capacità massima del buffer
- ➔ *_head* : indice della testa del buffer
- ➔ *_size* : per indicare la dimensione attuale del buffer

Per istruzioni del progetto, l'elemento più vecchio nel buffer è posizionato all'indice 0.
Inoltre, per evitare costosi spostamenti dei dati ad ogni inserimento o cancellazione, viene utilizzato un indice, *_head*, per memorizzare la cella contenente l'elemento più vecchio del buffer.
Dato che un array viene definito con dimensione fissa, ma nel quale gli elementi non nulli possono essere di numero inferiore alla dimensione massima, ho dovuto inserire nella classe il contatore *_size*, che memorizzerà il numero effettivo di elementi presenti. Tale contatore verrà incrementato ad ogni inserimento e decrementato ad ogni rimozione, mantenendolo comunque nel range da 0 a *_capacity* -1.

Infine si arriva al problema di gestione della circolarità di questo buffer. Dato che la testa si sposta, in seguito a inserimenti o cancellazioni, tramite una semplice equazione aritmetica posso calcolare la posizione di un elemento x del buffer con la formula:

$$x_n = (head + x) \% capacity$$

========== || Metodi implementati || ==========

La classe *cbuffer* possiede alcuni costruttori, che istanziano un oggetto *cbuffer* a partire da input diversi:

- ➔ Il costruttore di default, che non richiede parametri, per istanziare un cbuffer con capacità 0
- ➔ Un costruttore che prende in input la capacità voluta per il cbuffer da istanziare
- ➔ Un costruttore per copiare, che prende in input un altro cbuffer con elementi dello stesso tipo di quello da istanziare
- ➔ Un costruttore che, come da istruzioni del progetto, prende in input una coppia di iteratori di un cbuffer, i cui elementi sono di tipo diverso dal cbuffer già istanziato.

Inoltre, ho ridefinito alcuni operatori per un utilizzo più lineare con la mia classe cbuffer: innanzitutto ho definito un operatore di assegnamento, che agisce similmente al costruttore per copiare, poi ho ridefinito anche operatori di uguaglianza e disuguaglianza, i quali confronteranno i reference dei due cbuffer.
Altre funzioni che ho definito includono *begin* ed *end,* i quali istanzieranno l'iteratore di inizio e di fine sequenza, oppure funzioni utili per ottenere informazioni sul cbuffer quali capacità, dimensione attuale, indice della testa, indice della coda e restituzione dell'elemento a un dato indice in input.

Per istruzioni del progetto viene richiesta la possibilità di aggiungere elementi in coda al buffer o di rimuovere l'elemento in testa: per l'inserimento in coda il programma memorizza nella posizione di coda il valore da inserire, poi controlla se il buffer è pieno. Nel caso non ci sia spazio, viene spostata la testa del buffer in avanti, altrimenti viene incrementata la posizione del buffer.
Per rimuovere l'elemento in testa il programma verifica in primis che il buffer non sia vuoto, per evitare errori, tramite controllo sulla variabile *_size.* A controllo effettuato e superato, viene incrementata la posizione della testa e decrementata la dimensione attuale del buffer per rispecchiare la rimozione di un elemento in testa.

Inoltre, sempre per istruzioni del progetto, è necessario implementare l'accesso in lettura e scrittura dell'elemento i-esimo tramite operatore []. Tale implementazione viene fatta tramite ridefinizione dell'operatore [] nella classe cbuffer, definendo una funzione privata *l2f* (*logical-to-physical*), che a partire dall'indice a cui vuole accedere l'utente, ovvero l'indice che denomino "logico", restituisce l'indice della posizione reale nell'array di tale indice.

========== || Main || ==========

La funzione *main* di questo progetto si limita ad eseguire dei test, definiti nello stesso file, sul funzionamento di alcuni dei metodi implementati nel progetto.

Di nota tra le funzioni di test ho creato *test_metodi*, il quale istanzia vari cbuffer sfruttando i diversi costruttori e aggiungendo valori adatti a ciascuno.
Tra i vari test che ho inserito troviamo:

- ➔ Test degli operatori di assegnamento, di accesso e uso degli iteratori per l'accesso
- ➔ Test di istanziamento di due cbuffer, uno di tipo *int* e l'altro di tipo custom definito *person*.
  - ◆ Il tipo custom *person* è una *struct* con due campi string, per nome e cognome
- ➔ Test della funzione templace *check_if,* utilizzando tre predicati:
  - ◆ *positive_int* controlla che un dato intero sia maggiore o uguale a 0
  - ◆ *negative_int* controlla che un dato intero sia minore di 0
  - ◆ *surname_polo* controlla che il "cognome" dell'oggetto *person* sia "polo"

L'output dei test viene formattato in maniera leggibile e con relative informazioni su quale test viene eseguito.