Progetto: Buffer Circolare

Nicolas Vezzoli
n.vezzoli2@campus.unimib.it
Matricola 794545

1 Prefazione

Il progetto d'esame consiste nell'implementazione di un buffer circolare di elementi generici omogenei. Questa struttura dati utilizza un singolo buffer di dimensione fissa, e si distingue da un normale buffer per la contiguità esistente tra l'ultima e la prima locazione di memoria del buffer.

2 Implementazione del buffer circolare

Ho deciso di implementare il buffer circolare con un array. Questa struttura dati permette l'accesso alle singole celle di memoria in tempo costante ($O(1)$), garantendo quindi ottime prestazioni nell'accesso sia in lettura, che in scrittura. Il buffer è dichiarato come puntatore a tipo generico, deciso dal programma utilizzatore della classe template. Oltre a tale puntatore, ho dichiarato tre variabili intere senza segno:

- La capacità massima del buffer, ossia la dimensione dell'array;
- La dimensione (attuale) del buffer;
- L'indice cui si trova la testa del buffer.

Per migliorare la leggibilità del codice, ho usato l'istruzione `typedef` per rinominare il tipo `unsigned int` in `size_type`. Nelle specifiche del progetto si stabilisce che l'elemento più vecchio viene indicato con l'indice 0. Per non dover shiftare il contenuto del buffer dopo ogni inserimento o cancellazione, viene utilizzato un indiche che memorizza la cella che contiene l'elemento più vecchio dle buffer, ossia la testa.

Uno degli svantaggi dell'utilizzo di un array come struttura dati è quello di avere la sua dimensione, ossia il numero di elementi che può contenere, immutabile. Per tale ragione ho inserito nella classe il contatore size, che memorizza la dimensione attuale del buffer. Questo contatore indica il numero effettivo di elementi memorizzati nel buffer: viene incrementato quando viene inserito un nuovo elemento, viene decrementato quando un elemento viene rimosso. il valore è interno al range $(0,\ \mathit{capacità} - 1)$.

Per implementare la circolarità del buffer, ho utilizzato il seguente stratagemma: Poichè la testa si sposta in seguito a inserimenti e cancellazioni, necessito di un modo per sapere in che posizione si trovi effettivamente un determinato elemento. Servendomi dell'aritmetica modulare, è possibile calcolare la posizione nell'array di un elemento i del buffer con la seguente formula:

$$i_a = (testa + i)\%capacità \tag{1}$$

3 Metodi implementati

Nelle specifiche del progetto è richiesta la possibilità di aggiungere elementi in coda al buffer e di rimuovere elementi in testa. L'inserimento in coda al buffer memorizza nella posizione in coda al buffer il valore da inserire, dopodichè si paventano due possibilità: se il buffer è pieno, la testa del buffer viene spostata di una posizione in avanti, altrimenti viene incrementata la posizione del buffer.

La rimozione dalla testa del buffer verifica, innanzitutto, che il buffer non sia vuoto; per fare ciò, si verifica che la dimensione attuale del buffer sia diversa da 0 (la dimensione è memorizzata come intero senza segno, ergo non c'è la possibilità che assuma valore negativo). Se tale condizione è verificata, si incrementa la posizione della testa e si decrementa la dimensione attuale del buffer.

Altra specifica del progetto è l'accesso in lettura e scrittura al buffer tramite l'operatore []. Per poter soddisfare questa richiesta, ho dovuto ridefinire tale operatore per la mia classe `cbuffer`. A tal scopo ho definito una funzione membro privata chiamata `to_physical`, che prende in input l'indice "logico" della posizione cui vuole accedere l'utente, e ritorna la posizione reale all'interno dell'array. Ho definito due versioni di questo operatore: una permette l'acceso in sola lettura, mentre l'altra sia in lettura che in scrittura.

La classe cbuffer è dotata di alcuni costruttori, i quali instanziano un oggetto cbuffer a partire da diversi input:

- un costruttore di default, senza parametri, che instanzia un buffer di capacità 0;
- un costruttore secondario, che prende in input la capacità del buffer;
- un costruttore di copia, che prende in input un cbuffer i cui elementi sono dello stesso tipo del buffer instanziato;
- un costruttore specificatamente richiesto nelle specifiche, che prende in input una coppia di iteratori (inizio e fine) di un cbuffer, i cui elementi sono di tipo diverso rispetto al buffer instanziato. Come richiesto, la conversione dei tipi degli elementi dei due buffer è lasciata al compilatore.

Ho ridefinito alcuni operatori per permettere di essere usati con la mia classe cbuffer: in particolare ho ridefinito l'operatore di assegnamento, che, come il costruttore di copia, instanzia un nuovo cbuffer a partire da un altro cbuffer i cui elementi sono dello stesso tipo. Altri operatori ridefiniti sono quelli di uguaglianza e di disuguaglianza, che confrontano i reference di due cbuffer.

Ho definito anche le funzioni `begin` e `end`, nell'ottica di implementazione degli iteratori. Queste due funzioni membro istanziano, rispettivamente, l'iteratore di inizio e fine sequenza, richiamando il costruttore privato definito nella classe iterator, di cui la classe cbuffer è friend.

Infine, ho definito alcune funzioni membro che ritornano alcune informazioni sul buffer: la capacità, la dimensione attuale, l'indice cui si trova la testa, l'ìndice cui si trova la coda e una funzione che ritorna l'elemento che si trova alla posizione richiesta dal chiamante.

4 Iteratori

Viene richiesto che il buffer circolare supporti gli iteratori di lettura e scrittura. A tal scopo ho deciso di implementare sia la classe iterator che la classe const_iterator. Entrambe le classi sono annidate nella classe cbuffer. Poichè la richiesta si limita all'implementazione di iteratori di lettura e scrittura, ho deciso di mia iniziativa di implementare un forward iterator, poichè ho ritenuto sufficienti le operazioni supportate da questo tipo di iteratore. Questo anche a causa del fatto che gli elementi del buffer seguono una sorta di gerarchia dovuta alla loro differente anzianità (i più vecchi sono in prossimità della testa, i più nuovi sono in coda al buffer).

Oltre ad aver definito il costruttore di default e il costruttore di copia, ho definito un costruttore privato che prende in input un puntatore e il reference al relativo buffer. Questo costruttore viene utilizzato dalle funzioni `begin` e `begin` della classe cbuffer, per poter creare l'iteratore di inizio e fine sequenza.

5 Main

La funzione main si limita ad un semplice test di funzionamento di alcuni dei metodi implementati nel progetto.

Innanzitutto richiama la funzione `test_metodi_fondamentali`, che instanzia alcuni cbuffer con i diversi costruttori e aggiunge dei valori. Viene testato l'utilizzo degli operatori di assegnamento, di accesso (operatore []) e l'utilizzo degli iteratori per accedere agli elementi del buffer.

Terminata l'esecuzione della funzione vengono instanziati due buffer, uno di interi e uno di elementi di tipo person. Il tipo person è semplicemente uno struct dotato di due campi di tipo string: lo scopo è quello di testare la classe con tipi custom.

Fatto ciò, viene testata la funzione template `evaluate_if`. Questa funzione prende, come argomenti, un cbuffer e un predicato unario, e stampa sullo standard output, per ogni cella di memoria del buffer, se il predicato è soddisfatto o meno. Per testare questa funzione ho utilizzato tre predicati:

- `positive_int` verifica che, dato un intero, esso sia maggiore o uguale a 0;
- `negative_int` verifica che, dato un intero, esso sia strettamente minore di 0;

- `srn_ray` verifica che, data una variabile person, il cognome sia "Ray".

Questi predicati non corrispondono a casi particolarmente complessi da gestire, ma hanno lo scopo di mostrare il comportamento richiesto per la funzione `evaluate_if`.

6 Software utilizzati

Nello svolgimento del progetto, sono stati utilizzati i seguenti software, tutti gratuiti e con codice sorgente aperto:

- Editor di testo: Visual Studio Code 1.20.0;
- Compilatore: GNU GCC 6.3.0;
- Tool di autobuild: GNU Make 4.1;
- Tool di controllo memoria: Valgrind 3.12.0;
- Sistema operativo: Debian 9.3.