Programmazione ed Amministrazione di Sistema

Leonardo Fraquelli
l.fraquelli@campus.unimib.it
820651

June 3, 2019

Contents

Part I

Generazione di un SortedArray templato

Chapter 1

Introduzione

Dopo una attenta revisione del problema sono stato in grado di identificare i seguenti problemi:

1. Mantenimento della correttezza
   dei requisiti dell'array (ordine,dimensione)

2. Appropriata gestione della memoria
   ad ogni operazione

L'utilizzo di un semplice array templato per contenere i dati è sembrato immediato, non potendo utilizzare strutture dati della libreria standard. Una viabile alternativa sarebbe l'utilizzo di una ordered list per mantenere sempre (e più facilmente) ordinati i dati, questo avrebbe tuttavia potuto rallentare le operazioni di accesso ai dati.
Grosse difficolta sono state incontrate nella stesura di funzioni quali add/remove, durante le quali sono state [1] comprese correttamente le meccaniche dietro l'allocamento e il de-allocamento di dati.

[1] si suppone

Chapter 2

Dati

La struttura dati è templata per garantirne il corretto utilizzo con qualsiasi tipo di dato. L'array è organizzato come segue:

data è l'insieme di valori templati contenuti nell'array

size Indica la dimensione attuale (numero di dati)

equals è il funtore utilizzato per il confronto fra dati

order è il funtore utilizzato per ordinare i dati

Nella classe sono inoltre presenti due const-iteratori che iterano sui valori di data:

Random che può accedere ai dati nell'ordine desiderato,
La funzione begin restituisce il valore(tramite pointer) in posizione 0, la funzione end restituisce l'ultimo valore

Reverse un iteratore di tipo forward che può accedere ai dati solamente in ordine inverso,
La funzione begin restituisce il valore in posizione ultima, la funzione end restituisce il valore in posizione 0

Chapter 3

Scelte Implementative

L'implementazione della struttura dati ha richiesto:

1. Funzioni generiche per valori templati
2. Funzioni per l'aggiunta e la rimozione di dati, da cui segue la necessità di limitare queste operazioni dall'esterno della struttura dati
3. Funzioni generiche per iteratori templati

Da ciò deriva la scelta di utilizzare una classe; Sono quindi stati implementati i metodi fondamentali necessari:

Costruttore	inizializza un array vuoto
Copy Constructor	inizializza un array identico a quello in argomento
Distruttore	elimina le allocazioni in memoria attraverso una funzione **clear**
Assegnamento	permette di assegnare valori da un array a un altro

viene inoltre utilizzato un costruttore aggiuntivo per permettere di creare la classe da due iteratori generici.

Vengono inoltre utilizzate delle eccezioni utilizzando delle struct ed ereditando dalla gerarchia delle eccezioni standard per consentire una migliore identificazione di problemi.

Chapter 4

Funzioni

Sono state implementate le seguenti funzioni:

Update	Utilizzata sia per l'inizializzazione che l'aggiornamento di un array
Clear	Permette la corretta de-allocazione di un array
operator [ ]	che similmente a getElement consente di accedere a un elemento
add	che aggiunge un elemento all'array
remove	che rimuove un elemento dall'array
empty	che rimuove ogni elemento dall'array
filter	che rimuove ogni elemento dall'array che non rispetti un funtore *filtro* e ne genera uno nuovo
getElement	che permette di accedere a un elemento
getSize	che restituisce la dimensione dell'array
Exists	che restituisce l'esistenza di un elemento
swap	che è una reimplementazione di std::Swap

Chapter 5

Main

Il main del progetto è utilizzato unicamente a scopo di test.
I test sono svolti su tutte le funzioni elencate nel capitolo precedente.
Vengono testati valori di tipo:
Int,*Char*,*Voce*[1]
Per ogni valore sono stati definiti funtori di Uguaglianza,Ordine,Filtraggio.
Il progetto non presenta leak o errori su Virtual-Box con OS: "Ubuntu 18.04.2 LTS"

[1]E' stata reimplementata da un precedente progetto,Rubrica, con l'aggiunta degli operatori $>$,$<$ per eseguirne l'ordinamento