PROGETTO C++

AUTORE: Cattaneo Alessandro

MATRICOLA: 851612

E-MAIL: a.cattaneo81@campus.unimib.it

RICHIESTA PROGETTO

Il progetto richiedeva l'implementazione di un set di dati, di tipo generico, non ordinato e senza elementi duplicati.

Questa classe può essere vista come una variante della classe set già presente nella libreria standard c++ (std::Set), con l'unica differenza che i set proposti dalla libreria sono anche ordinati.

STRUTTURA UTILIZZATA

Per la realizzazione del set è indispensabile creare una struttura dati di supporto che permetta di incapsulasse le informazioni dei singoli elementi, e un predicato generico per confrontare elementi.

La scelta dell'utilizzo del funtore, piuttosto che la ridefinizione dell'operatore ==, è stata fatta su ispirazione della classe std::Set che utilizza un funtore di confronto per verificare se 2 elementi sono uguali oltre che al loro ordinamento.

Come struttura dati di supporto è stata scelta un **nodo** (node) così composta:

1. Un campo **value** che incapsula un qualsiasi valore assegnato al nodo
2. Un puntatore **next** a un nodo che punta al nodo successivo, se presente, rispetto a quello corrente.
   Questo puntatore risulta essere necessario in quanto il set non ha una dimensione nota a priori e i vari node non sono salvati in modo contiguo in memoria

Per quando riguarda il set questo è composto da:

1. Un puntatore **head** che punta al primo nodo del set e servirà come punto d'accesso al set
2. Un attributo **setSize** che conterrà la lunghezza corrente del set
3. Un predicato **equal** che è il predicato di uguaglianza mediante il quale verranno confrontati i valori dei nodi del set

SCELTE IMPLEMENTATIVE

Come la maggior parte delle strutture dati indicizzate, anche in questo caso, gli indici attribuiti ai singoli elementi vanno da 0 a N-1, mentre la lunghezza va da 0 a N.

Inoltre avendo solo un puntatore all'elemento successivo di un determinato nodo e non vedendo la necessità di scorrimento multi direzionale del set, ho optato per l'utilizzo di un forward iterator costante.

ULTERIORI DETTAGLI IMPLEMENTATIVI

Per la realizzazione del set oltre ai metodi fondamentali sono state implementate le seguenti funzioni:

- **Add:** inserisce un elemento nel set, per verificare che non ci siano duplicati controlla elemento per elemento partendo dell'elemento in testa fino alla fine del set ed è per questo motivo che l'inserimento è stato effettuato in coda.
  Nel caso in cui l'add fallisse il set non verrà modificato, il nodo creato eliminato e l'utente verrà informato dell'accaduto
- **Remove:** rimuove l'elemento dal set che ha come valore il valore del parametro passato alla funzione come parametro, se si cerca di eliminare un elemento non presente la funzione non farà nulla, inoltre per funzionare correttamente il set non deve essere vuoto altrimenti l'asserzione nella funzione fallirà.
- **Operatore <<:** permette di stampare il contenuto del set.
  Operatore []: permette di accedere all'i-esimo elemento del set, può lanciare un'eccezione out_of_range (eccezione standard di C++) se l'indice va oltre la lunghezza del set.
- **Operatore ==:** verifica che 2 set siano uguali, due set vengono considerati uguali se hanno la stessa lunghezza e gli stessi elementi a prescindere dall'ordine
- **getSetSize()** che è una funzione che restituisce la lunghezza corrente del set.
  Lo scopo di questa funzione è principalmente di debug che però per completezza della classe reputavo necessaria.
- **costruttore avente come parametri due iteratori**: crea un Set a partire da una sequenza di dati usando una coppia di iteratori generici.
- **emptySet()** che è una funzione di supporto al distruttore che serve per cancellare i vari nodi del set settare la dimensione del set a 0 e il puntatore _head a nullptr.
  Risulta inoltre comoda nella gestione degli errori legati alla funzione di add in quanto la new in essa contenuta potrebbe fallire e quindi per mantenere la classe in uno stato coerente i set, per i quali l'aggiunta fallisce, vengono svuotati ed eliminati

Nel progetto sono state anche implementate 3 funzioni globali **filter_out, operartor+** e **operator-** il cui compito è evidente dal testo del progetto e quindi sarebbe ridondante inserirlo nella documentazione

Per quanto riguarda il ritorno delle funzioni globali ho scelto di ritornare i set per copia per i seguenti motivi:

- è impossibile ritornare il set per reference in quanto lo scope del set è limitato alla funzione in cui viene creato e il reference essendo sullo stack muore alla fine dell'esecuzione della funzione
- ritornare il set per puntatore invece viola il pattern RAII in quanto cederei le proprietà dell'oggetto e la responsabilità dell'eliminazione non sarebbe più del distruttore della classe set ma dovrei chiamare una delete dal main

REQUISITI FUNZIONAMENTO

Nel caso la classe venga utilizzata su dati primitivi è necessario solo definire un predicato di uguaglianza tra essi, per evitare l'inserimento di elementi duplicati, e un predicato booleano per la funzione globale filter_out.

Mentre per tipi di dato custom oltre ai predicati citati sopra è necessario ridefinire anche l'operatore di stream <<

TEST

Il debugging del codice è stato fatto funzione per funzione (anche con valgrind) nel corso della stesura del codice, e l'utilizzo delle stampe al posto delle asserzioni è un'abitudine personale.

In particolare la classe è stata testata su interi, e su due tipi di dati custom:

- **punti 2D** composti quindi da coordinate X e Y intere
- **persone** composte da 2 stringhe (nome e cognome) ed un intero (età)

Spero di aver testato il codice in tutte le sue funzionalità e in tutti i suoi casi limite, ovviamente i test sono racchiusi in delle funzioni specifiche che poi vengono invocate nel metodo main.

Mi auguro che le oltre 2000 righe di test abbiano coperto tutte le funzionalità e non diano brutte sorprese