Programmazione e Amministrazione di Sistema

Relazione Progetto C++ del 12/02/2019

Nome: Francesco
Cognome: Stranieri
Matricola: 816551
Email: f.stranieri1@campus.unimib.it

Introduzione

Il progetto richiedeva la progettazione e la realizzazione di una classe che implementava un Set di elementi generici T.
Un Set è una collezione di dati che non può contenere duplicati.

Implementazione Set e Tipi di Dati

Dopo aver valutato attentamente il problema ho deciso di rappresentare gli elementi del Set appoggiandomi ad una lista concatenata. Ogni elemento è quindi rappresentato da un nodo della lista, il quale contiene il valore dell'elemento, e un puntatore all'elemento successivo del Set, se presente.

La classe possiede inoltre un puntatore al primo elemento del Set, un contatore che tiene traccia del numero di elementi presenti al suo interno e un funtore di uguaglianza, il cui uso verrà illustrato nel corso della relazione.

Inizialmente avevo optato per una rappresentazione attraverso un albero binario di ricerca. Questo permetteva di ottimizzare i tempi e di avere gli elementi ordinati ma, poiché la traccia non richiedeva questa particolare specifica, ho deciso di utilizzare la lista.

Nel dettaglio, un nodo della lista è rappresentato tramite una *struct* contenente:

- Value, di tipo T, che rappresenta l'effettivo valore dell'elemento del Set. Essendo la classe templata, il tipo di dato viene passato come parametro e di conseguenza la classe, e le relative funzioni, devono essere generiche permettendo così il riutilizzo del codice.
- Next, puntatore all'elemento successivo del Set, se presente, o a NIL, alternativamente.

Poiché la classe è templata è necessario introdurre anche un funtore di uguaglianza per permettere di identificare quando due elementi del Set sono uguali (necessario poiché, ad esempio, il metodo impiegato per verificare se due numeri Interi sono uguali è diverso dal metodo usato per verificare se due Stringhe sono uguali).

Per quanto riguarda gli iteratori, la classe è stata dotata di *iteratori forward* costanti. La scelta mi è sembrata in linea con l'implementazione della classe e sufficiente per l'utilizzo.

Metodi Fondamentali

Abbiamo inizialmente implementato i metodi fondamentali per la classe.

Il ***Costruttore di default***, inizializza un Set vuoto.

Il ***Costruttore secondario***, permette di creare un Set a partire da una sequenza di dati definita da una coppia generica di iteratori, passati come parametro.

Il ***Costruttore di copia***, permette di istanziare un Set con i valori presi da un altro Set, passato come parametro.

Il ***Distruttore***, permette di rimuovere la memoria allocata da un Set durante l'esecuzione del programma. Esso si appoggia al metodo *clear* che verrà illustrato successivamente.

Metodi Implementati

In ordine di apparizione all'interno del file "set.h" troviamo il metodo ***count*** il quale permette di conoscere il numero di elementi presenti nel Set. Tale numero viene incrementato ogni qualvolta viene aggiunto un elemento nel Set e decrementato quando, all'opposto, viene rimosso.

Il metodo ***clear*** rimuove tutti gli elementi presenti nel Set, deallocando la memoria occupata da ogni singolo elemento. Poiché ad ogni *new* nel metodo *add* deve corrispondere una *delete*, questo metodo è fondamentale per far si che non si presentino memory leak.

Il metodo ***add*** permette l'aggiunta di un elemento nel Set; il valore dell'elemento da aggiungere viene passato come parametro costante.
Per l'implementazione di tale metodo ho deciso di appoggiarmi al metodo *find* in modo tale che, se l'elemento candidato ad essere aggiunto nel Set è già presente, lancio una eccezione senza dover invocare la *new* ed allocare conseguentemente memoria.

L’allocazione di memoria solo se strettamente necessaria rende il programma più robusto ad eventuali memory leak.
L’aggiunta di un elemento viene eseguita in coda (per un incremento delle prestazioni si può ricorrere all’inserimento in testa).

Un caso particolare dell’*add* è l’aggiunta di un elemento in un Set vuoto; in tal caso è necessario aggiornare il puntatore del primo elemento del Set all’elemento appena aggiunto.

Il metodo ***remove*** permette la rimozione di un elemento dal Set. Come già visto nella *add*, è stato previsto di appoggiarsi al metodo *find* e di lanciare una eccezione nel caso in cui l’elemento non sia presente nel Set.

Abbiamo due casi particolari di cui bisogna tener conto:

- Se si rimuove un elemento dal Set, ed esso è l’unico elemento presente, allora il Set diventerà vuoto.
- Se invece si rimuove il primo elemento del Set, allora il puntatore alla testa del Set dovrà essere aggiornato di conseguenza.

Per identificare l’elemento da rimuovere è necessario l’uso del funtore di uguaglianza che confronta il valore dell’elemento passato come parametro (costante) al valore di ogni elemento del Set.

Il metodo ***find*** permette di ricercare un elemento all’interno del Set. Scorrendo il Set, esso restituisce il valore booleano TRUE se l’elemento è presente, FALSE altrimenti.

Anche in questo caso, per identificare l’elemento da ricercare è necessario l’uso del funtore di uguaglianza.

La funzione globale e generica ***filter_out*** permette di filtrare da uno specifico Set, passato come parametro, tutti gli elementi che

non soddisfano un dato predicato, passato anch'esso come parametro. Per ogni elemento del Set, se esso non soddisfa il predicato allora verrà aggiunto, tramite la funzione *add*, nel Set risultante.

Ridefinizione degli Operatori

Ho deciso di effettuare l'overloading degli Operatori descritti qui di seguito.

L'*Operatore di assegnamento* (**=**) permette la copia tra Set, usando il Costruttore di copia e la funzione *swap* della libreria *std*.

L'*Operatore di accesso* (**[ ]**) permette di accedere, esclusivamente in lettura come specificato dalla traccia, ad uno specifico elemento del Set, tramite un indice passato come parametro. Per permettere l'accesso solo in lettura il metodo è stato dichiarato *const*.
Anche se non richiesto esplicitamente ho deciso di implementare una eccezione nel caso in cui l'indice non fosse coerente con la dimensione del Set (ad esempio se maggiore al numero di elementi attualmente presenti al suo interno).

L'*Operatore di stampa* (**<<**) permette di spedire sullo stream di output il contenuto del Set. Essendo il Set composto da elementi visualizzeremo il relativo valore. Per una scelta puramente estetica ho deciso di visualizzare gli elementi tra parentesi graffe.

L'*Operatore di concatenazione* (**+**) permette di restituire un Set, contenente gli elementi dei due Set passati come parametro. Ho implementato questa funzione scorrendo a turno i due Set passati in input e aggiungendo, tramite il metodo *add*, ogni elemento presente nei due Set. Avendo già implementato con l'*add* il controllo sugli elementi duplicati, sappiamo che se l'operazione di concatenazione andrà a buon fine il Set risultante avrà esclusivamente elementi non duplicati.

Eccezioni

Per un corretto e coerente funzionamento della classe è stato necessario introdurre delle eccezioni.

L'***add_exception*** viene lanciata quando si tenta di inserire un elemento che è già presente nel Set. È stata implementata per il metodo *add* e di conseguenza in tutti i metodi che si appoggiano ad esso.

La ***remove_exception*** viene lanciata quando si tenta di rimuovere un elemento che non è presente nel Set. È stata implementata per il metodo *remove*.

L'***index_exception*** viene lanciata quando si tenta di accedere ad un elemento del Set tramite un indice che non è coerente con il numero di elementi presenti nel Set. È stata implementata per l'*Operatore di accesso*.

Main

Il ***main*** è stato immaginato in modo tale da non prevedere alcuna iterazione con l'utente.

Il primo tipo di dato che ho pensato di testare riguarda i numeri Interi.

Innanzitutto è stato necessario definire il funtore di uguaglianza e i predicati necessari per la funzione *filter_out*. Nel dettaglio, i predicati scelti sono stati *is_odd* ed *is_even*.

Nella prima parte vengono testati i diversi costruttori implementati, quindi *di default*, *secondario* e *di copia*, provando anche alcuni casi particolari come ad esempio l'invocazione degli stessi su Set vuoti. In seguito è stata testata l'aggiunta di elementi e la rimozione degli stessi, tenendo in considerazione i casi particolari relativi a queste due funzioni e affrontati nel corso della relazione, generando anche le relative eccezioni. Successivamente sono stati testati gli operatori *di assegnamento* e *di accesso* in lettura, generando anche la relativa *index_exception*. Per verificare che l'operatore *di accesso* fosse abilitato esclusivamente in lettura si è tentato di accedere in scrittura ad un elemento del Set; questa operazione generava giustamente un errore e per questo motivo è stata commentata.

Nella seconda parte viene invece testata la funzione *filter_out*. Partendo da un Set contenente i numeri interi da 1 a 9, sono stati filtrati da esso i numeri pari e i numeri dispari in due distinti Set . Tramite l'*operatore di concatenazione*, tra i due Set ottenuti dall'operazione di filtraggio, si è poi ritornati al Set di partenza.

Infine lo stesso flusso di operazioni è stato eseguito su oggetti di tipo String e di tipo Voce mostrando così che la classe generica Set, e i metodi presenti al suo interno, sono stati implementati in modo tale da poter accettare qualsiasi tipo di dato in input.