Programmazione e Amministrazione di Sistema

Davide Marchetti

Università degli Studi di Milano Bicocca
d.marchetti8@campus.unimib.it
Matricola: 815990

1 Introduzione

Per l'implementazione del progetto è richiesta l'implementazione di una classe Set di elementi generici T, la quale non può contenere elementi duplicati.

2 Implementazione

2.1 Scelte d'implementazione

La struttura dati scelta per l'implementazione consiste in una lista dinamica composta da nodi, definiti nel seguente modo:

```
struct node {
    /**
     * Valore dell'elemento, essendo contenuto in una classe templata
     * il tipo di dato sara' definito dall'utilizzatore
     */
    T value;

    /**
     * Puntatore all'elemento successivo della lista. Se il nodo e'
     * l'ultimo della lista, next punta a null(0)
     */
    node* next;
};
```

2.2 Classe generica Set

Si è scelto di implementare la classe Set come classe generica. Per un corretto utilizzo, si deve specificare il tipo base di ogni elemento e un funtore di uguaglianza utilizzato per garantire l'unicità degli elementi all'interno del set.

Se quest'ultimo viene istanziato su tipi base C++, il funtore di uguaglianza è facoltativo, in quanto verrà utilizzato quello della libreria standard relativo al tipo T utilizzato.

Questa dichiarazione può essere riassunta come segue:

```
/**
 * Set data structure implementation with no duplicated data allowed
 *
 * @tparam T value type
 * @tparam std::equal_to<T> equal functor for items comparsion
 */
template <typename T, typename Equal = std::equal_to<T>>
class set {
```

2.3 Variabili interne alla classe

La classe contiene al suo interno tre proprietà private per gestire internamente i dati:

```
/**
 * Puntatore alla testa della lista
 */
node *_head;

/**
 * Conteggio degli elementi attualmente inseriti nella lista.
 */
unsigned int _count;

/**
 * Funtore di uguaglianza fornito dall'utilizzatore/della classe
     standard.
 */
Equal _equal;
```

2.4 Operazioni disponibili

Nella classe sono stati implementati i metodi essenziali per il corretto funzionamento della stessa, quali: costruttori (default, copia, coppia di iteratori), distruttore e operatore di assegnamento.
Inoltre, come da specifiche, per interagire con il set sono disponibili i seguenti metodi:

Add Aggiunge un elemento in coda alla lista, se quest'ultimo è già presente viene generata un'eccezione di tipo value_already_exists[1].

```
void add(const T &value);
```

[1] Vedi paragrafo 2.6

Remove Rimuove l'elemento specificato dal set, se quest'ultimo NON è presente, viene generata un'eccezione di tipo value_does_not_exists[2].

```
void remove(const T &value);
```

Operator[] Ritorna l'elemento presente nella posizione *index* richiesta.
Per la verifica di un corretto accesso si è scelto di utilizzare una *assert* e lasciare i dovuti controlli all'utilizzatore della classe.

```
T operator[](int index) const;
```

Contains Verifica, tramite il funtore *_equal*, se l'elemento richiesto è presente nel set. Ritorna *true* se l'elemento è presente, *false* altrimenti.

```
bool contains(const T &value) const;
```

Clear Rimuove tutti gli elementi dal set, chiamando la *delete* su ciascuno, evitando memory leak. Questa funzione è utilizzata anche dal distruttore della classe.

```
void clear();
```

Count Restituisce il numero di elementi contenuti nel set.

```
unsigned int count() const;
```

2.5 Iteratori

La specifica richiedeva anche l'implementazione di un *const_iterator*. Il suo utilizzo è possibile attraverso i metodi:

Begin Ritorna un'iteratore che punta al primo dato inserito nel set.

```
const_iterator begin() const;
```

End Ritorna un'iteratore che punta alla fine del set (cioè a *nil*).

```
const_iterator end() const;
```

[2] Vedi paragrafo 2.6

2.6 Eccezioni

Le funzioni di aggiunta e rimozione richiedevano il lancio di eccezioni sotto determinate condizioni espresse qui di seguito:

value_already_exists Eccezione lanciata se si tenta di inserire un elemento già presente nel set.

```
class value_already_exists {};
```

value_does_not_exist Eccezione lanciata se si cerca di rimuovere un elemento NON presente nel set.

```
class value_does_not_exist {};
```

2.7 Funzioni globali

La specifica del progetto si concludeva specificando la definizione di alcune funzioni globali operanti sulla classe generica Set, oltre all'implementazione dell'operatore di stream per permetterne la stampa del contenuto tramite standard output.

Filter Out Riceve in input un reference ad un Set di dati generici T e un funtore di uguaglianza E, oltre ad un funtore esterno P che specifica la condizione secondo la quale verranno presi gli elementi dal set dato in input.

```
template <typename T, typename E, typename P>
set<T, E> filter_out(const set<T, E> &s, P condition);
```

Operator+ Riceve in input due reference a due Set di dati generici T con relativo funtore di uguaglianza E, e crea un nuovo set dato dalla concatenazione di *s1* ed *s2*. Per la concatenazione viene creato un nuovo Set temporaneo, sfruttando il costruttore di copia; successivamente vengono accodati gli altri elementi sfruttando gli iteratori e la funzione add.

```
template <typename T, typename E>
set<T, E> operator+(const set<T, E> &s1, const set<T, E> &s2);
```

Operator« Lavorando su un Set generico, questa funzione deve essere anch'essa templata con tipo di dato T e funtore di uguaglianza E. La stampa viene eseguita utilizzando il *const_iterator* dichiarato all'interno della classe.

```
template <typename T, typename E>
std::ostream &operator<<(std::ostream &stream, const set<T, E> &s);
```

3 Main

Il main è composto dalla dichiarazione di diverse struct utili al testing delle varie parti del programma e da tre funzioni ciascuna per testare diverse tipologie di input.

Si parte da dei **test fondamentali** su un semplice set di interi, con funtore della libreria standard.

Successivamente viene utilizzata una struct *voce* rieseguendo le stesse operazioni su un set dichiarato come set di elementi voce con funtore dichiarato nel main.

```
set<voce, voce_equals> s
```

Infine viene effettuato un rapido test su una struttura dati più complessa, definita come un set di set di interi e funtore anch'esso dichiarato nel main.

```
set<set<int>, set_equal> multi_set
```