Labo 6 – Implementazione di dizionari con alberi binari di ricerca

Scopo

- Implementare il TDD Dizionario con chiave di tipo string e valore di tipo string, usando alberi binari di ricerca, seguendo le indicazioni viste in classe, ossia
  - Le operazioni devono essere implementate con funzioni/procedure ricorsive che operino in theta(h) nel caso peggiore, con h=altezza dell'albero
  - In ogni nodo non si può tenere traccia del padre, ossia divieto di aggiunta del puntatore al padre alla struct che rappresenta il nodo

Scopo

- Il TDD Dizionario così implementato deve
  - Fornire tutte le operazioni standard, ossia

```
Error insertElem(const Key, const Value, Dictionary&);
Error deleteElem(const Key, Dictionary&);
Value search(const Key, const Dictionary&);
Dictionary createEmptyDict();
```

  - Poter essere utilizzato nel main fornito per il Laboratorio 5, estendendo in modo opportune il file header dictionary.h e compilando in modo adeguato

- Sarà necessario inoltre implementare le funzioni di lettura da file e stdin (saranno molto simili a quelle del Labo 5) nonché la funzione di stampa corrispondente ad una visita **DFS simmetrica**

Materiale del Labo 5

- Dictionary-main.cpp
  - il main contiene il menu che permette di eseguire le varie operazioni sul dizionario
- Dictionary.h
  - Contiene strutture dati e prototipi delle funzioni che andranno implementate nel file dictionary-hashtable.cpp e richiamate in Dictionary-main.cpp
  - Header - NON MODIFICARE
- Dictionary-hashtable.cpp
  - Implementazione del Dizionario mediante tabelle di hash
  - Implementare qui le 6 funzioni richieste!
- StringUtility.cpp, StringUtility.h
  - Questi file contengono delle funzioni per “nomalizzare” le chiavi, rendendo tutti i caratteri minuscoli ed eliminando gli spazi
- Folder EngEng
  - Contiene dei file ‘dizionario’ chiave-valore

Come procedere

- Scaricate il codice fornito per svolgere il Laboratorio 5, mantenete solo una delle possibili implementazioni fornite (ovvero, fate un po' di pulizia cancellando i file che non vi serviranno) e partite da un header e un file ausiliario già implementati: sarà molto più facile modificare del codice in cui trovate già gli include corretti, i prototipi delle funzioni da implementare, etc..., piuttosto che partire da zero! Date nomi “sensati” ai file nuovi.

Come procedere

- Implementate per prime la funzione insertElem e la funzione print. Non potete fare alcun test se non avete inserito elementi nel dizionario, e non potete vedere se sono stati inseriti correttamente se non chiamate una print.
- Potete testare la insertElem usando il main così come è e sfruttando l'opzione “c” per inserire una coppia (chiave, valore) alla volta: inserite una coppia e fate una stampa; inseritene un'altra e fate un'altra stampa...
- Non tentate di implementare la lettura da file finché non siete più che certi che la insertElem (con inserimento della chiave e valore da tastiera) funzioni correttamente!!!

Come procedere

- Implementate la search dopo avere implementato la lettura da file, in modo da poterla testare su dizionari più complessi
- La deleteElem è la funzione più difficile: implementatela per ultima, quando tutte le alter sono state testate in modo accurato. Seguite lo pseudo-codice del testo Aho-Hopcroft-Ullman

Come procedere

- In fase di sviluppo del codice è naturalmente possibile - anzi, è consigliato - modificare il main in modo da verificare separatamente le singole funzionalità implementate; in alternativa, è possibile creare un main nuovo nel quale vengono effettuate chiamate di funzioni ad hoc per il testing. Tuttavia alla fine il programma deve funzionare con il main fornito originariamente nel Laboratorio 5
- Potete inoltre aggiungere tutte le funzioni ausiliarie che vi servono. Attenzione al fatto che non faranno parte del TDD!

Compilazione

```
namespace dict {

// Codici di errore

enum Error {OK, FAIL};

// Tipi e costanti

const int tableDim = 1000; // da modificare per fare esperimenti diversi

typedef string Key;        // tipo base
typedef string Value;      // tipo base

const Key emptyKey = "###RESERVED KEYWORD### EMPTY KEY";
const Value emptyValue = "###RESERVED KEYWORD### EMPTY VALUE";

typedef struct {
    Key key;
    Value value;
} Elem;


#ifdef USE_HASH_TABLE
// Implementazione basata su tabella hash

struct cell;
typedef cell* Bucket;
const Bucket emptyBucket = NULL;
typedef Bucket* Dictionary; // un Dictionary è un array di dimensione tableDim (che
viene fissata in fase di inizializzazione), di puntatori a cell (che abbiamo chiamato
"Bucket"); come già fatto in altre occasioni, la struttura di cell è definita nel file ausiliario
per incapsulare il più possibile l'informazione

#endif
```

...

Cosa consegnare

- Il main deve essere consegnato così come è stato distribuito per il Laboratorio 5. Il comando per compilare correttamente il programma dovrà essere esplicitamente indicato da qualche parte (nel main, oppure in un README) o contenuto in un Makefile.
- La consegna di questo laboratorio deve includere anche il foglio excel dei tempi di esecuzione delle singole operazioni già predisposta per il Laboratorio 5, compilata in ogni sua parte incluse quelle richieste dal Laboratorio 5 ma comprendente anche le misure di prestazione relative all'albero binario di ricerca. Se non si è riusciti a implementare la tabella di hash con liste di collisione come richiesto dal Laboratorio 5, è possibile utilizzare la soluzione messa a disposizione dai docenti per completare il foglio excel.