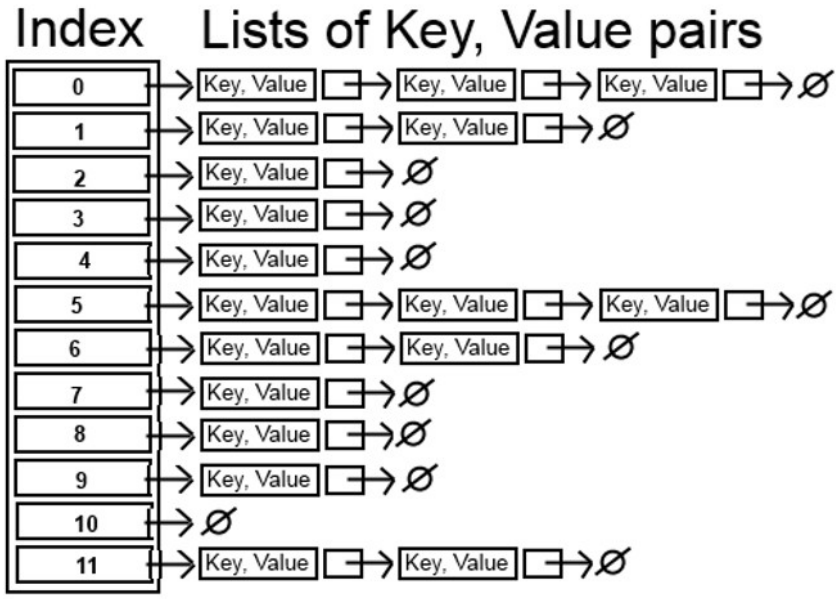
LABORATORIO 5

Algoritmi e strutture dati

DIZIONARIO

- Mediante la struttura dati "**tabella di hash con liste di collisione**"




```
Abasia : (n.) Inability to coordinate muscular actions properly in walking.
Abgeordnetenhaus : (n.) See Legislature, Austria, Prussia.
Abra : (n.) A narrow pass or defile; a break in a mesa.
Abreaction : (n.) See Catharsis, below.
```

FUNZIONI DA IMPLEMENTARE

h1(), funzione di hash che considera unicamente il valore ascii del primo carattere della chiave e restituisce il resto della divisione di tale valore per tableDim;

h2(), funzione di hash che somma il codice ascii di ogni carattere nella chiave restituisce il resto della divisione di tale somma per tableDim;

h3(), funzione che dovete inventare voi seguendo qualche criterio ragionevole e chemotiverete nel file di comprensione dei dati sperimentali;

insertElem();

deleteElem();

search();

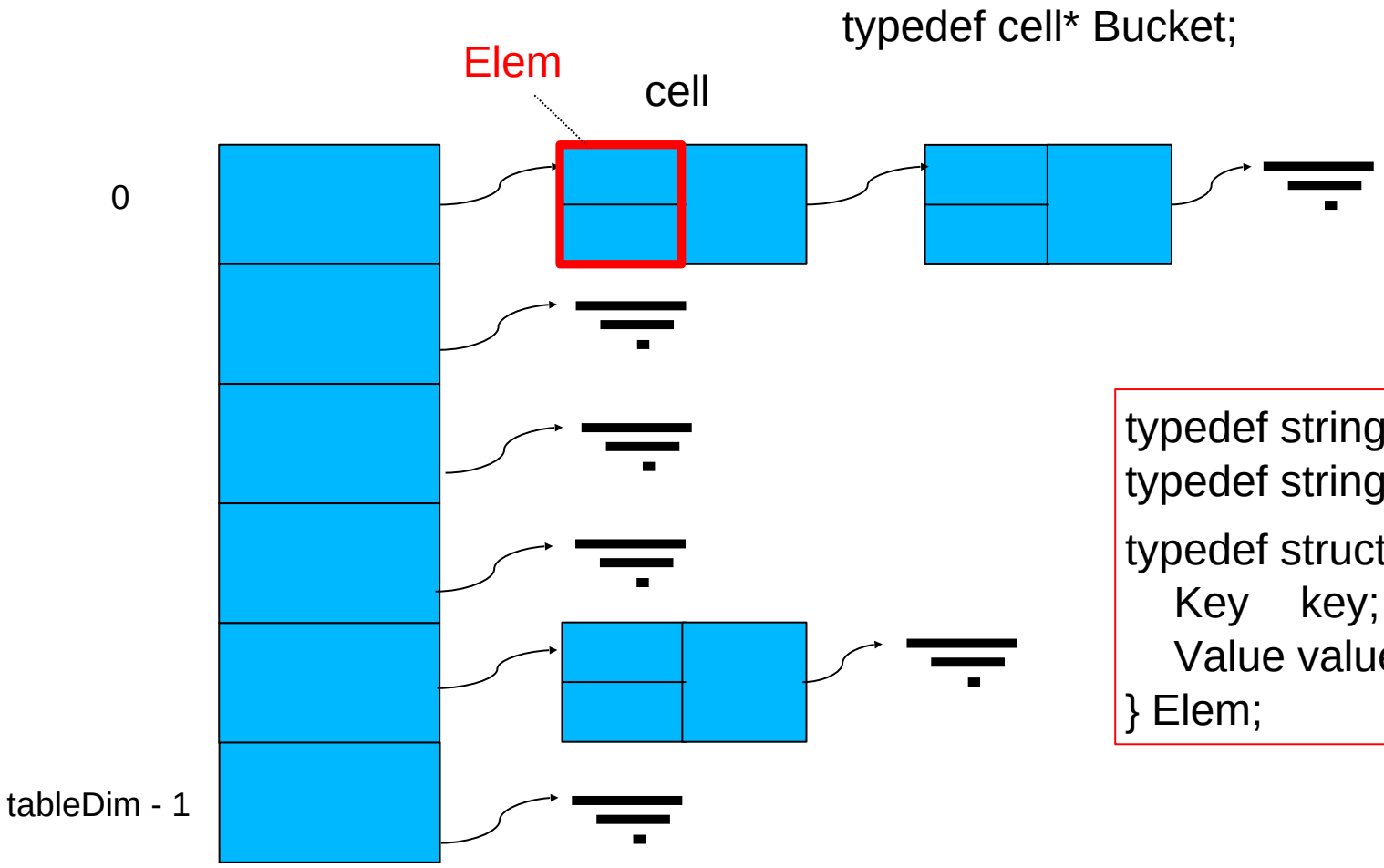
dictionary-hashtable.cpp

VALORE ASCII

```
// librerie utilizzate
#include <stdio.h>
#include <stdlib.h>

main() {
// ricavo il codice ASCII di una lettera
int A_IN_ASCII = (int)('A');
printf("La lettera A in codice ASCII e': %d\n\n", A_IN_ASCII);

// ricavo una lettera a partire da un codice ASCII
char carattere = (char)(65);
printf("Il codice ASCII 65 e' lettera: %c\n\n", carattere);

system("pause");
}
```

int('A');

IMPLEMENTAZIONE HASH TABLE




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

Dictionary (array dinamico)

nullptr

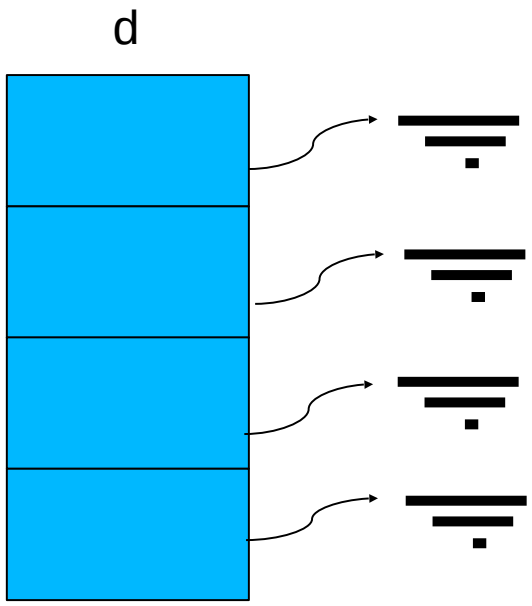
File .zip (TRACCIA)

Composta da ***diversi*** *file + file di input*

- **Dictionary-main.cpp**
  - il main contiene il menu che permette di eseguire le varie operazioni sul dizionario - **NON MODIFICARE**
- **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

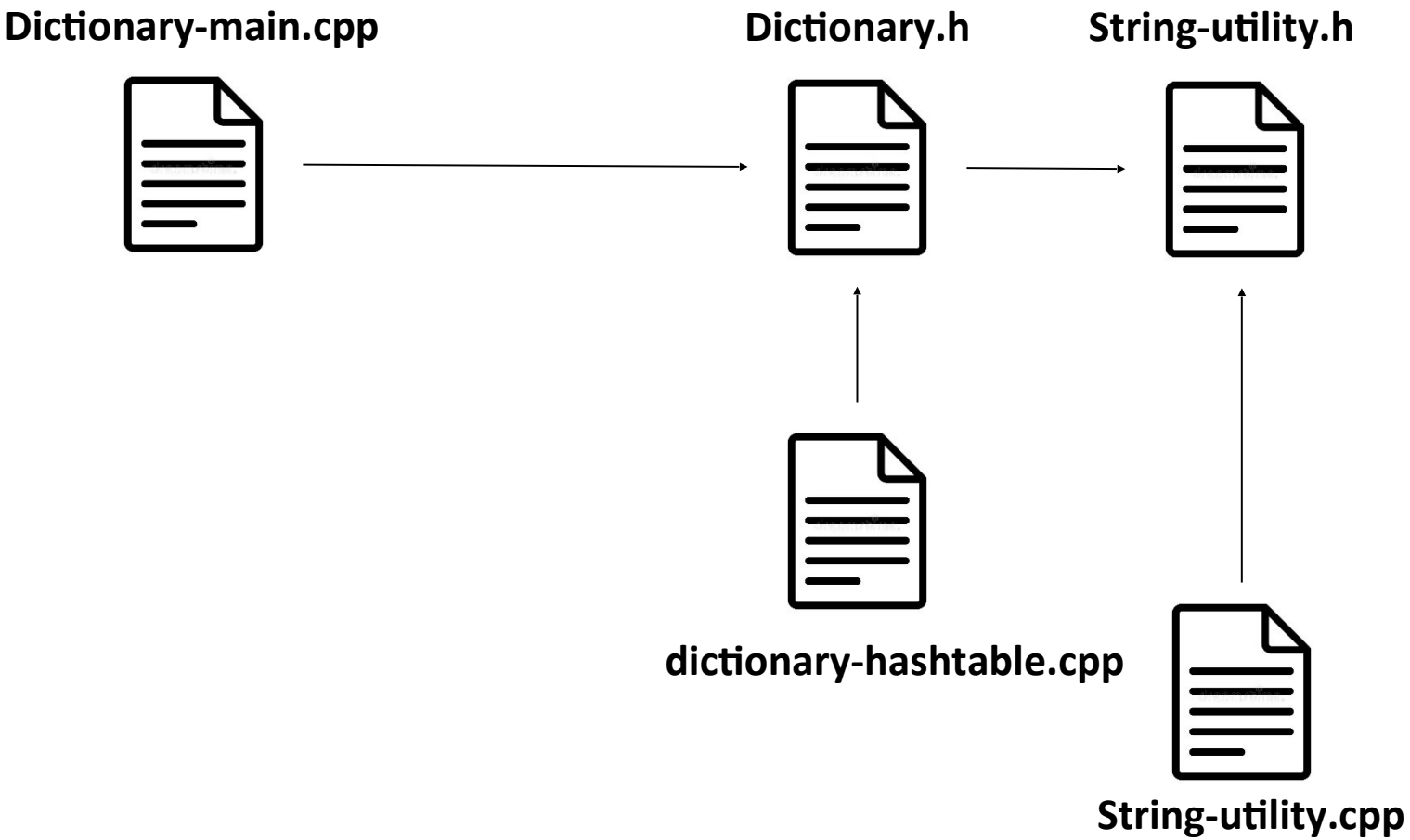
createEmptyDict()

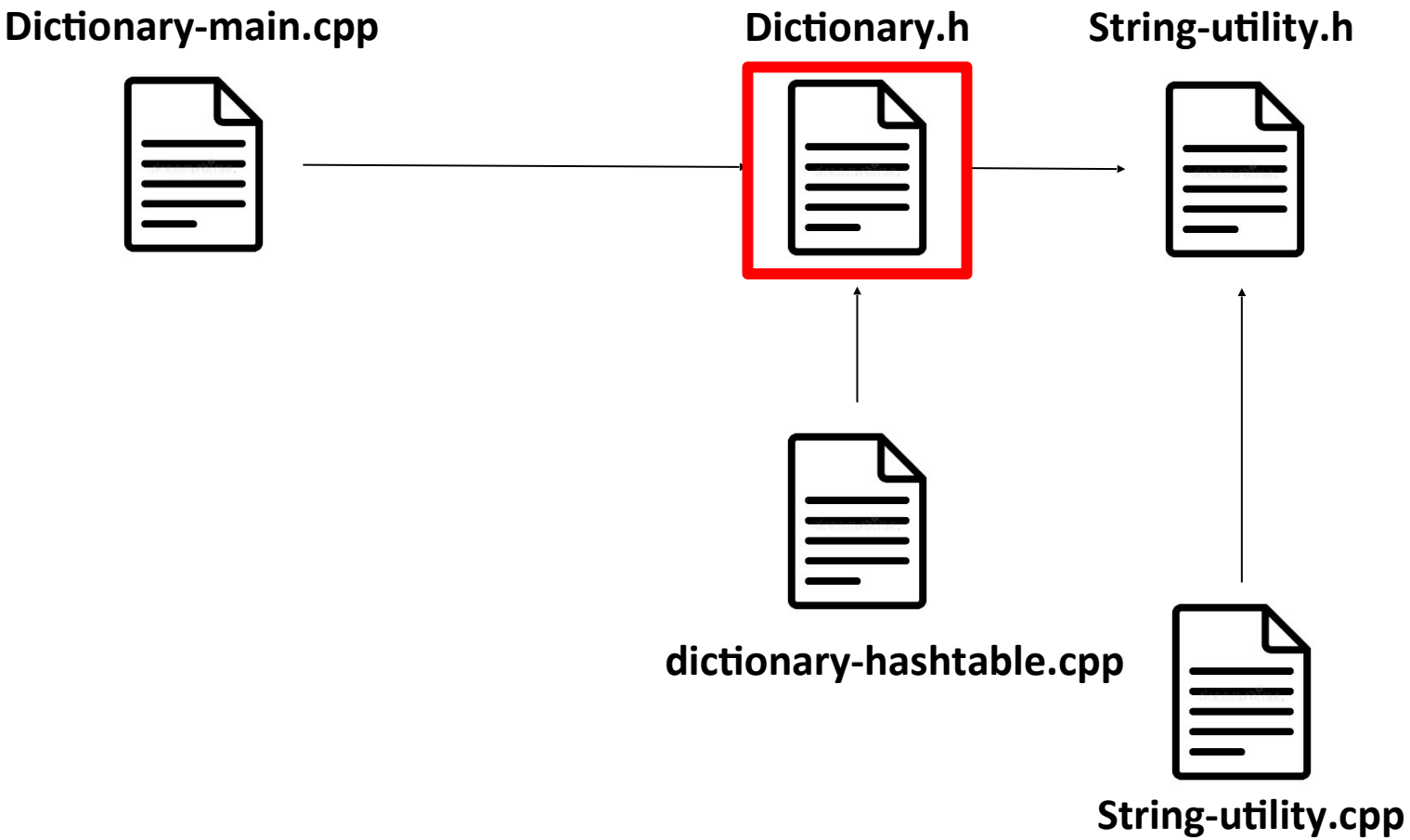
```
Dictionary dict::createEmptyDict()
{
    Bucket* d = new Bucket[tableDim];
    for (int i=0; i < tableDim; ++i)
       d[i]=emptyBucket;
    return d;
}
```

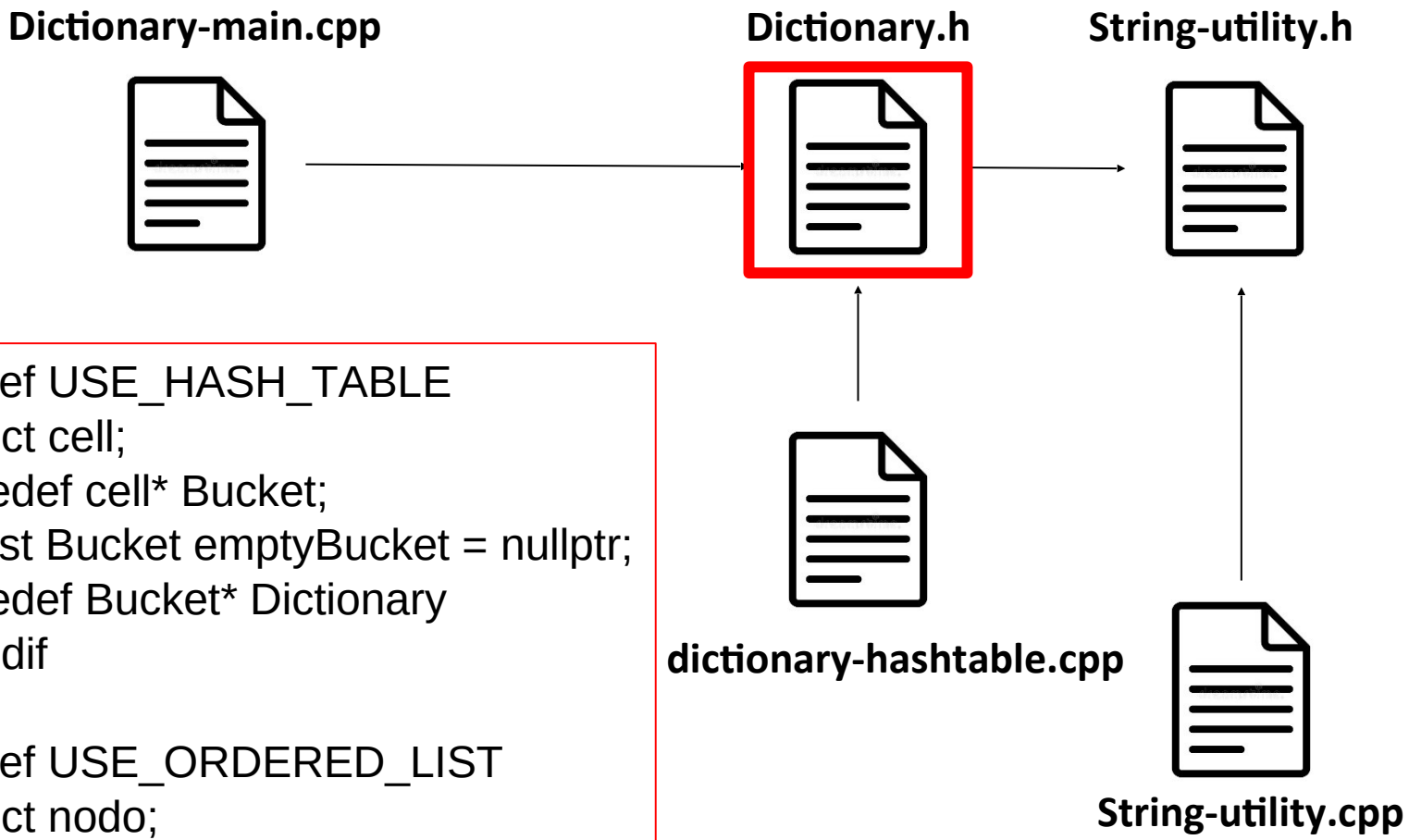


Dictionary.h

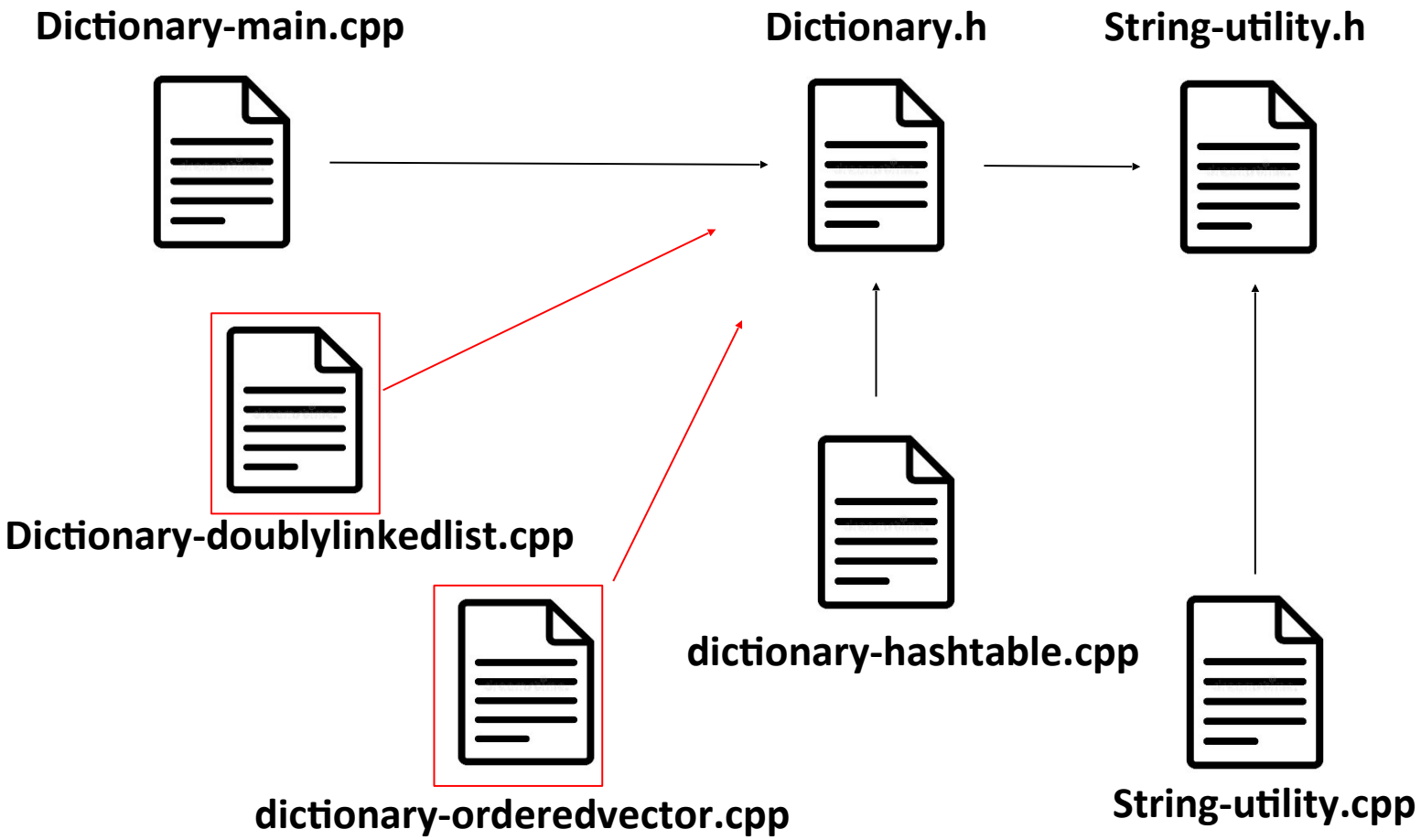
```
const int tableDim = 1000;
typedef cell* Bucket;
const Bucket emptyBucket = nullptr;
```


Dictionary-main.cpp
Dictionary.h
String-utility.h
dictionary-hashtable.cpp
String-utility.cpp


Dictionary-main.cpp
Dictionary.h
String-utility.h
dictionary-hashtable.cpp
String-utility.cpp




```
#ifdef USE_HASH_TABLE
struct cell;
typedef cell* Bucket;
const Bucket emptyBucket = nullptr;
typedef Bucket* Dictionary
#endif

#ifdef USE_ORDERED_LIST
struct nodo;
typedef nodo* Dictionary;
#endif

#ifdef USE_ORDERED_VECTOR
typedef vector<Elem> Dictionary;
#endif
```


Dictionary-main.cpp
Dictionary.h
String-utility.h
Dictionary-doublylinkedlist.cpp
dictionary-hashtable.cpp
dictionary-orderedvector.cpp
String-utility.cpp

organizzazione-hash-table.xls

SPERIMENTAZIONE (1)

	# elementi memorizzati	# bucket	# a
Test1.1: ascii.txt, tableDim = 20, hash h1			
Test1.2: ascii.txt, tableDim = 100, hash h1			
Test1.3: ascii.txt, tableDim = 1000, hash h1			
Test2.1: ascii.txt, tableDim = 20, hash h2			
Test2.2: ascii.txt, tableDim = 100, hash h2			
Test2.3: ascii.txt, tableDim = 1000, hash h2			
Test3.1: ascii.txt, tableDim = 20, hash h3			
Test3.2: ascii.txt, tableDim = 100, hash h3			
Test3.3: ascii.txt, tableDim = 1000, hash h3			
Test4.1: d.txt, tableDim = 20, hash h1			
Test4.2: d.txt, tableDim = 100, hash h1			
Test4.3: d.txt, tableDim = 1000, hash h1			
Test5.1: d.txt, tableDim = 20, hash h2			
Test5.2: d.txt, tableDim = 100, hash h2			
Test5.3: d.txt, tableDim = 1000, hash h2			
Test6.1: d.txt, tableDim = 20, hash h3			
Test6.2: d.txt, tableDim = 100, hash h3			
Test6.3: d.txt, tableDim = 1000, hash h3			

tempi-di-esecuzione-operazioni-dict.xls

SPERIMENTAZIONE (2)

	a.txt, lettura da file	a.txt, inserimento (“asas”, “prova”)
Test1: vector ordinato		
Test2: liste collegate ordinate		
Test3: hash table con dim 20, h1		
Test4: hash table con dim 100, h1		
Test5: hash table con dim 1000, h1		
Test6: hash table con dim 20, h2		
Test7: hash table con dim 100, h2		
Test8: hash table con dim 1000, h2		
Test9: hash table con dim 20, h3		
Test10: hash table con dim 100, h3		
Test11: hash table con dim 1000, h3		

ComprendiamoGliEsperimenti.doc

‘QUESTIONARIO’

Comprendiamo i dati sperimentali

Organizzazione della tabella di hash

1. Quando si memorizzano i dati presenti nel file ascii.txt in una tabella di dimensione 20, la deviazione standard che otteniamo usando h1 e h2 è la stessa. Analogamente, se la tabella è di dimensione 100 la deviazione standard risulta la stessa, a prescindere dal fatto che si sia usata h1 o h2. Lo stesso accade se la dimensione è 1000. Per quale ragione?

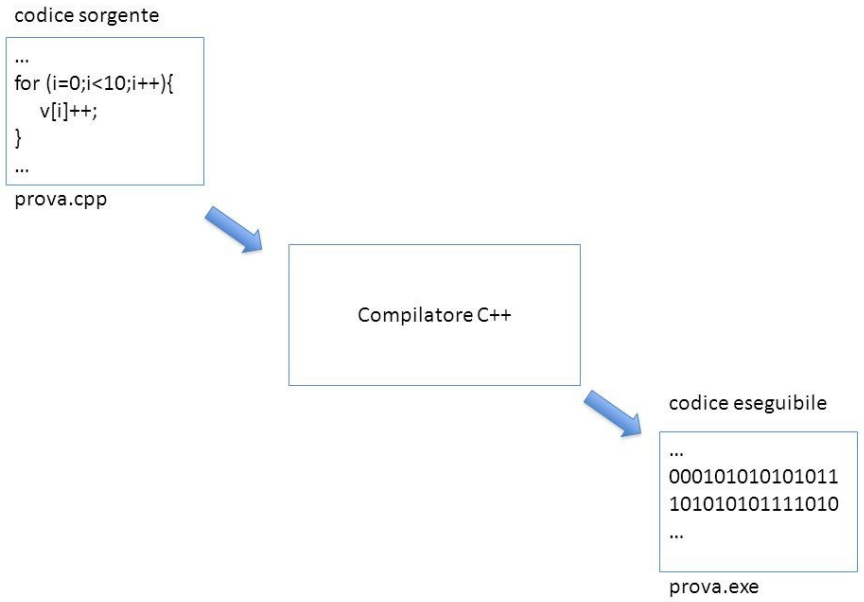
2. A differenza di quanto accade con i dati nel file ascii.txt, quando si memorizzano i dati presenti nel file d.txt la deviazione standard varia sia con la dimensione della tabella, sia con il variare della funzione di hash usata. Per quale ragione?

3. Qual è la funzione di hash migliore, tra h1 e h2, per memorizzare dati come quelli nei file a.txt, b.txt, etc? Per quale ragione?

Efficienza delle operazioni del TDD “Dizionario” usando strutture dati diverse

1. Perché ricercare “azure” in una tabella di hash in cui sono stati inseriti i dati nel file a.txt risulta sempre un'operazione molto efficiente, a prescindere dalle caratteristiche (dimensione, funzione di hash usata) della tabella stessa?

COMPILAZIONE ED ESECUZIONE



./a.out

1. g++ -std=c++11 -Wall –DUSE_HASH_TABLE dictionary-hashtable.cpp string-utility.cpp dictionary-main.cpp
2. g++ -std=c++11 -Wall –DUSE_ORDERED_VECTOR dictionary-orderedvector.cpp string-utility.cpp dictionary-main.cpp
3. g++ -std=c++11 -Wall –DUSE_ORDERED_LIST dictionary-doublylinkedlist.cpp string-utility.cpp dictionary-main.cpp