Roll No : 1911034

Batch : A3

Exp no: 9

Grade :

**Title: Implementation of Maps and Dictionaries**

**Objective:** To implement different operations on dictionary – size, find\_element, findall\_elements, remove\_elements, remove\_allelements.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 3 | To describe concepts of advance data structures like Map and Dictionaries |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –*Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –*Jean Paul Tremblay,

Paul G. Sorenson

1. *Data Structures A Pseudo Approach with C –*Richard F. Gilberg & Behrouz A. Forouzan
2. [*https://www.geeksforgeeks.org/sorting-algorithms/*](https://www.geeksforgeeks.org/sorting-algorithms/)
3. [*https://www.tutorialspoint.com/data\_structures\_algorithms/sorting\_algorithms.htm*](https://www.tutorialspoint.com/data_structures_algorithms/sorting_algorithms.htm)

**Dictionary Data Structure:**

Dictionary is an abstract data structure that supports the following operations:

search(K key) (returns the value associated with the given key)

insert(K key, V value)

delete(K key)

Each element stored in a dictionary is identifed by a key of type K. Dictionary represents a mapping from keys to values.

**Dictionaries have numerous applications**

Search can return a special value if key is absent in dictionary

**Examples**

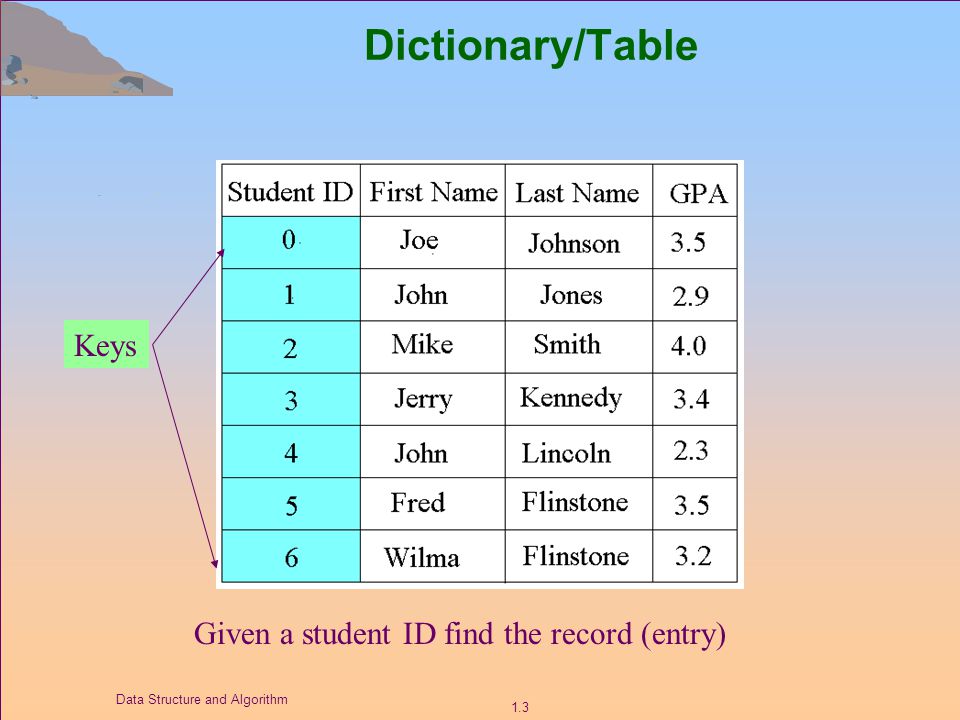
contact book key: name of person; value: telephone number

table of program variable identifiers key: identifier; value: address in memory

property-value collection key: property name; value: associated value

natural language dictionary key: word in language X; value: word in language Y etc.

Example of a Dictionary representing a table that displays the ID no., first name and last name along with the CGPA of a student.

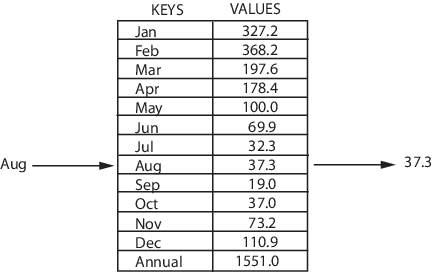


**Map Data Structure**

A Map is a type of fast key lookup data structure that offers a flexible means of indexing into its individual elements. Unlike most array data structures in the MATLAB® software that only allow access to the elements by means of integer indices, the indices for a Map can be nearly any scalar numeric value or a character vector.

Indices into the elements of a Map are called keys. These keys, along with the data values associated with them, are stored within the Map. Each entry of a Map contains exactly one unique key and its corresponding value. Indexing into the Map of rainfall statistics shown below with a character vector representing the month of August yields the value internally associated with that month, 37.3.

**Mean monthly rainfall statistics (mm)**



Keys are not restricted to integers as they are with other arrays. Specifically, a key may be any of the following types:

* 1-by-N character array
* Scalar real double or single
* Signed or unsigned scalar integer

The values stored in a Map can be of any type. This includes arrays of numeric values, structures, cells, character arrays, objects, or other Maps.

**Dictionary ADT**

**Operations**

* **Dictionary create()**  
  creates empty dictionary
* **boolean isEmpty(Dictionary d)**  
  tells whether the dictionary **d**is empty
* **put(Dictionary d, Key k, Value v)**  
  associates key **k** with a value **v;**  
  if key **k**already presents in the dictionary  
  old value is replaced by **v**
* **Value get(Dictionary d, Key** **k)**  
  returns a value, associated with key **k**  
  or null, if dictionary contains no such key
* **remove(Dictionary d, Key** **k)**  
  removes key **k** and associated value
* **destroy(Dictionary d)**  
  destroys dictionary**d**

**Difference between Dictionary and Map**

1. Maps are a subset of dictionary.
2. Dictionary is defined  as having the insert, delete, and find functions.
3. Map as used by Java is a dictionary with the requirement that keys mapping to values are strictly mapped as a one-to-one function.
4. A dictionary might have more than one key map to one value, or one key map to several values (like chaining in a hasthtable), eg Twitter hashtag searches.

**Applications of Map:**

All C++ map are used to link a key to a value. So if you have key of something link some data item value a map is the ideal container to use. C++ have several type of maps: map, multimap, unordered map, and unordered multimap.

A map has a pair of key-value and every key has only 1 value and the pairs are sorted by the key with some comparison function

A multimap has a pairs of key-value and every key can have multiple values and the pairs are sorted by the key with some comparison function

An unordered map has a pair of key-values where every key has only 1 value and the key is placed based on a hash function. A poor hash function defined by every key generating the same hash value can turn the map into a list or vector as it

An unordered multimap has a pair of key-value and every key can have multiple values and the key is placed on a hash function.

**ADT Map:**

**Map Operations**

* **Construct a map**

Default constructor

Copy constructor

Construct a map from a range of elements

* **Test a map**

Test for emptiness: empty()

* **Add one or more elements to a map**

insert(element): Insert the element passed and return a pair struct containing the iterator position of that element (or of the previously existing element) and a boolean indicating whether the insertion succeeded

insert(iter, element): Insert the element passed, with iter a "hint" at the location (for efficiency purposes) and return the iterator position of the inserted element (or of the previously existing element)

insert(startIter, endIter): Insert elements from the given range, but return nothing

* **Remove one or more elements from a map**

clear():Remove all elements

erase(keyVal): Remove the element(s) with key(s) equal to keyVal

erase(iter): Remove element at the iterator position

erase(startIter, endIter): Remove all elements in the given range

* **Retrieve value of a map element**

mapName[keyVal]: Return value corresponding to a given key

* **Search operations**

count(keyVal): Return number of elements (0 or 1) having the given key value

find(keyVal): Return iterator position of element whose key is same as keyVal, or end()

lower\_bound(keyVal): Return first iterator position where keyVal would be inserted (position of first element with key value >= keyVal)

upper\_bound(keyVal): Return last iterator position where keyVal would be inserted (position of first element with key value > keyVal)

equal\_range(keyVal): Return the values of lower\_bound(keyVal) and upper\_bound(keyVal) as a pair struct, with lower\_bound first and upper\_bound second

**ADT Multimap:**

Multimap Operations:

insert(element):Insert the element passed and return the iterator position of that element

count(keyVal):Return number of elements having the given key value

**Implementation of Dictionary in C++**

#include <iostream>

#include <map>

#include <iterator>

using namespace std;

int main()

{

std:: multimap <string,map <string,int> > Weather;

int n, op;

cout << "Enter Number of Entries: ";

cin >> n;

for(int i=0;i<n;i++)

{

string city;

string date;

int temp;

cout << "Entry " << i+1 << endl;

cout << "Enter City: ";

cin >> city;

cout << "Enter Date: ";

cin >> date;

cout << "Enter Temperature: ";

cin >> temp;

std:: map <string,int> Wd;

Wd.emplace(date,temp);

Weather.emplace(city,Wd);

}

std::multimap<string, map<string,int> >::iterator it;

std::map<string,int>::iterator it1;

string city1;

string date1;

string city2;

int max = 0;

menu:;

cout << "\n";

cout << "Menu: " << endl;

cout << "1. Display all Data Recorded" << endl;

cout << "2. Searching Data Based on City & Date" << endl;

cout << "3. To Find Maximum Recorded Temperature of a city" << endl;

cout << "4. Deleting Record of a city" << endl;

cout << "5. Deleting Particular Record of City " << endl;

cout << "6. End" << endl;

cout << "Enter Option: ";

cin >> op;

switch(op)

{

case 1:

system("clear");

cout << "\nData is:\n";

cout << "City\t\tDate\t\tTemperature" << endl;

for(it=Weather.begin();it != Weather.end();it++)

{

for(it1 = it -> second.begin();it1 != it -> second.end();it1++)

{

cout << it -> first << "\t\t";

cout << it1 -> first << "\t\t";

cout << it1 -> second << "\t";

cout << endl;

}

}

goto menu;

case 2:

system("clear");

cout << "Enter City: ";

cin >> city1;

cout << "Enter Date: ";

cin >> date1;

for(it=Weather.begin();it != Weather.end();it++)

{

if(it -> first == city1)

{

for(it1 = it->second.begin();it1 != it->second.end();it1++)

{

if(it1->first == date1)

{

cout << it -> first << " ";

cout << it1 -> first << " ";

cout << it1 -> second << " ";

}

}

cout << endl;

}

}

goto menu;

case 3:

system("clear");

max = 0;

cout << "Enter City: ";

cin >> city2;

for(it=Weather.begin();it != Weather.end();it++)

{

if(it -> first == city2)

{

for(it1 = it->second.begin();it1 != it->second.end();it1++)

{

if(it1 -> second > max)

{

max = it1 -> second;

}

}

cout << endl;

}

}

cout<< "Maximum Temperature Recorded for "<<city2<<" is "<<max;

goto menu;

case 4:

system("clear");

max = 0;

cout << "Enter City: ";

cin >> city2;

Weather.erase(city2);

cout << " Record of " << city2 << " Deleted "<< endl;

goto menu;

case 5:

system("clear");

cout << "Enter City: ";

cin >> city1;

cout << "Enter Date: ";

cin >> date1;

for(it=Weather.begin();it != Weather.end();it++)

{

if(it -> first == city1)

{

for(it1 = it->second.begin();it1 != it->second.end();it1++)

{

if(it1->first == date1)

{

(it->second).erase(date1);

}

}

cout << endl;

}

}

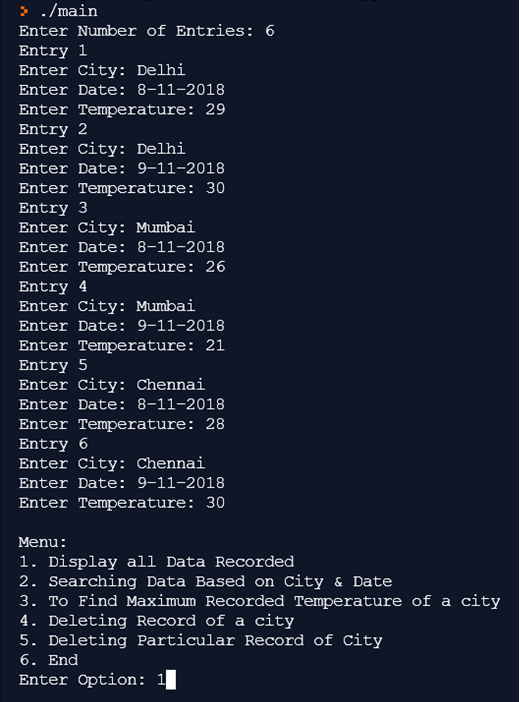
goto menu;

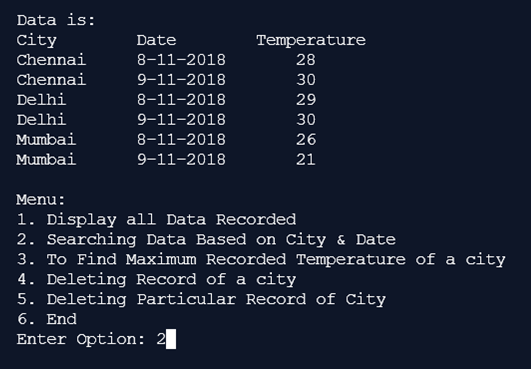
case 6:

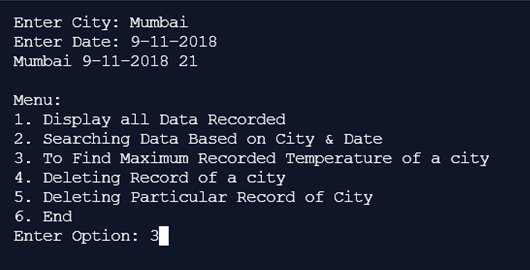
break;

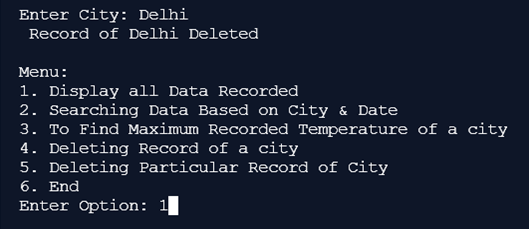
}

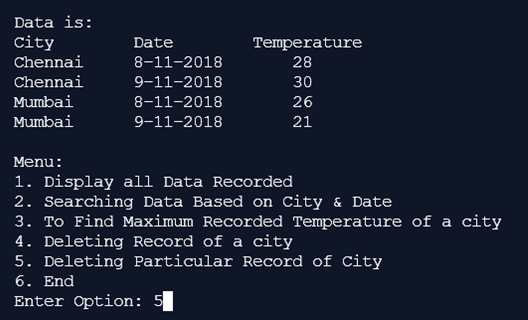
return 0;

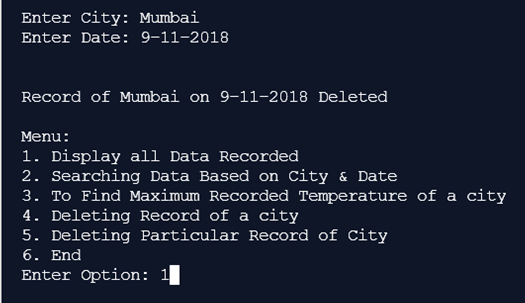
****

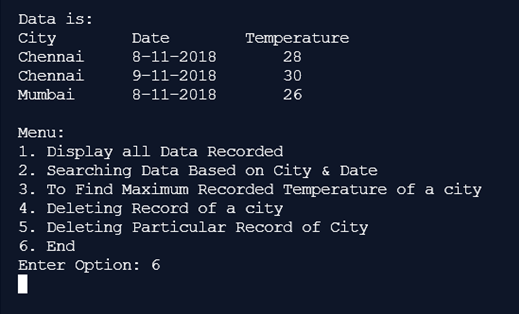
****

****

****

****

****

****

**Conclusion : In this study experiment , we were able to explore and understand the concepts of advance data structures such as maps and dictionaries.**