

**Information Technology University, Lahore**

**Department of Computer Science**

***CS243: Data Structures and Algorithms***

**Fall 2020**

|  |  |
| --- | --- |
| ***Course Instructor: Miss Momina Azam*** | ***Dated: 7 Nov, 2020*** |
| ***Lab Engineer: Aqsa Khalid*** | ***Semester: 3*** |
| ***Session: 2019-2023*** | ***Batch: BSCS2019*** |

**Lab 6. Implementing an open addressing hash table**

**Due: 7 November, 2020**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll No** | **Lab Marks / 25** | **Viva Marks / 5** | **Total Marks / 30** |
| Muhammad Muneeb Ur Rahman | BSCS19057 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Instructions:**

* Make sure to read instructions carefully before attempting questions.
* Each question carries 5 marks.
* Make separate code files for each question and name them like “**Q1.cpp**”,”**Q2.cpp**” etc. The code for respective questions should also be copied in the space provided.
* Make a word file for all theoretical questions.
* Write your comments on methodology and results. Mere reporting results is not useful. **Insightful comments carry marks**.
* You are allowed to discuss with fellow students and with TA for general advice, but you must submit **your** own work.
* Plagiarism will not be tolerated and will be dealt according to “**Academic Code of Conduct**”.
* We will not just be looking for correct answers rather higher grades will be awarded to solutions that demonstrate a clear understanding of the material. Style matters and will be a factor in the grade.
* All code files and this manual should be in a folder. Name this folder with your roll number e.g. **Lab1\_BSCS19001**. Upload the zipped folder in *Google Classroom*. **We do not receive labs through email**.
* Late submission is subject to very valid reasons for example serious illness or tragedy.

**TASKS**

**TASK 1: ( Due 11:59 am )**

You are to program an open addressing hash table. Use starting array size to be 110.

Your hash table must resolve collisions by placing the key at the next available empty slot. The Hash Table must hold the key value pairs given in the file two-letter-words. This file holds key value pairs: two letter words as key and a meaning that is separated by a tab character.

1. What would be the struct you will define for your Hash Table array?
2. What must be the members of the struct?
3. You are to program the Hash Table with following basic operations:

* Insert
* Search [20 Marks]

|  |
| --- |
| #include <iostream>  #include <string>  #include <fstream>  using namespace std;  #define A 54059 /\* a prime \*/  #define B 76963 /\* another prime \*/  #define C 86969 /\* yet another prime \*/  #define FIRSTH 37 /\* also prime \*/  int hash2(string word)  // POST: the index of entry is returned  {  int sum = 0;  for (int k = 0; k < word.length(); k++)  sum = sum + int(word[k]);  return sum % 110;  }  int hash3(string word)  {  int seed = 131;  unsigned long hash = 0;  for (int i = 0; i < word.length(); i++)  {  hash = (hash \* seed) + word[i];  }  return hash % 110;  }  struct ppair {  string key;  string value;  ppair(string k, string v) {  key = k;  value = v;  }  };  struct HashTable {  size\_t size;  ppair\*\* data;  };  void additem(string key, string value, HashTable& T) {  int index = hash2(key);  //Loop Find the EMPTY Space or check whether key is already found,  //If no empty space is found, the value is inserted at the end.  while (index<110 && T.data[index] != nullptr && T.data[index][0].key != key)  ++index;  if (index == 110)--index;  if (T.data[index] == nullptr)  T.data[index] = new ppair(key, value);  else  T.data[index][0].value = value;  }  //Search The Index..  int search(string key, HashTable& T) {  int index = hash2(key);  while (index<110 && T.data[index] != nullptr) {  if (T.data[index][0].key == key)  return index;  else  ++index;  }  return -1;  }  //Check the value if it is found...  void testfunction(HashTable& T) {  ifstream rdr("test.txt");  for (int i = 0; i < 110; ++i) {  string key;  getline(rdr, key, '\n');  if (search(key, T))  cout << "Found" << endl;  else  cout << "Not Found" << endl;  }  }  int main() {  HashTable T1;  T1.size = 110;  T1.data = new ppair \* [T1.size];  for (int i = 0; i < T1.size; ++i)  T1.data[i] = nullptr;  string key;  string value;  ifstream rdr("data.txt");  for (int i = 0; i < T1.size; ++i) {  getline(rdr, key, '\t');  getline(rdr, value, '\n');  cout << key << " " << value << endl;  additem(key, value, T1);  }  testfunction(T1);  return 0;  } |

**TASK 2: ( Due 11:59 pm )**

Use the solution for Task 1 & array size greater than 110, and give an analytical report with performance figures you gather from repeatedly searching different keys, especially with regard to the lookup function as that is the most important function in a hash table.

The analysis must be based on two criterions:

1. Hash Table size

2. Quality of the Hash Function ... you are provided three different hash functions to work with

(in the file named “hash-functions.cpp”). [10 Marks]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Size Of Data Entry: 103**  **Size Of Array Initialized with nullptr in given in the column heading**   1. With **hash2** function :  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | **Case 1 (Size = 50)** | **Case 2 (Size = 103)** | **Case 2 (Size = 120)** | **Case 3 (Size = 150)** | | Collision Rate : | 78/103 | 74/103 | 76/103 | 76 |  1. With **hash3** function :  |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | **Case 1 (Size = 50)** | **Case 2 (Size = 103)** | **Case 2 (Size = 120)** | **Case 3 (Size = 150)** | | Collision Rate : | 74/103 | 50/103 | 41/103 | 24/103 | |
| **CODE**  #include <iostream>  #include <string>  #include <fstream>  using namespace std;  #define A 54059 /\* a prime \*/  #define B 76963 /\* another prime \*/  #define C 86969 /\* yet another prime \*/  #define FIRSTH 37 /\* also prime \*/  const int SIZE = 150;  int hash2(string word)  // POST: the index of entry is returned  {  int sum = 0;  for (int k = 0; k < word.length(); k++)  sum = sum + int(word[k]);  return sum % SIZE;  }  int hash3(string word)  {  int seed = 131;  unsigned long hash = 0;  for (int i = 0; i < word.length(); i++)  {  hash = (hash \* seed) + word[i];  }  return hash % SIZE;  }  struct ppair {  string key;  string value;  ppair(string k, string v) {  key = k;  value = v;  }  };    struct HashTable {  size\_t size;  ppair\*\* data;  };  void additem(string key, string value, HashTable& T, int& collision) {  int index = hash2(key);  //Loop Find the EMPTY Space or check whether key is already found,  //If no empty space is found, the value is inserted at the end.  if (index < SIZE && T.data[index] != nullptr && T.data[index][0].key != key) {  ++collision;  }  while (index < SIZE && T.data[index] != nullptr && T.data[index][0].key != key) {  ++index;  }  if (index == SIZE)--index;  if (T.data[index] == nullptr)  T.data[index] = new ppair(key, value);  else  T.data[index][0].value = value;  }  //Search The Index..  int search(string key, HashTable& T) {  int index = hash2(key);  while (index<SIZE && T.data[index] != nullptr) {  if (T.data[index][0].key == key)  return index;  else  ++index;  }  return -1;  }  //Check the value if it is found...  void testfunction(HashTable& T) {  ifstream rdr("test.txt");  for (int i = 0; i < 103; ++i) {  string key;  getline(rdr, key, '\n');  if (search(key, T)!=-1)  cout << "Found" << endl;  else  cout << "Not Found" << endl;  }  }  int main() {  HashTable T1;  T1.size = SIZE;  int collisions = 0;  T1.data = new ppair \* [T1.size];  for (int i = 0; i < T1.size; ++i)  T1.data[i] = nullptr;  string key;  string value;  ifstream rdr("data.txt");  for (int i = 0; i < 103; ++i) {  getline(rdr, key, '\t');  getline(rdr, value, '\n');  cout << key << " " << value << endl;  if (i == 6)  cout << endl;  additem(key, value, T1, collisions);  }  cout << endl << endl << collisions << endl << endl;  testfunction(T1);  return 0;  } |
| **COMMENT**  The highlighted part of the code is used to find the collisions when a value is added to the array. The variable **collision** in the function additem increments if the value is already found on the **index** returned by the hash function |
| **CONCLUSION**  From the values found it can be concluded that hash3 performs better than hash2 for a specific size, and as the size increase from 50 to150 the spread is also better with the hash3 function. |