Name:- Chad Huntebrinker

Answer the following Questions

**Q1: [5 Points]** What would the Huffman code look like if all symbols in the alphabet had an equal frequency? In this case, how would the Huffman tree look like (what type of binary tree you would have)? What would play a major rule in deciding the length of the code/codes? Justify your answers.

|  |
| --- |
| Use this space to answer the question  For a Huffman tree, you cannot have any letters as an internal node. In other words, all the letters in the alphabet would have to be leaves or external nodes. So, you would get a tree where all the characters are at the bottom of the tree. Some of the characters would be a level above the bottom of the tree because there would not be enough characters to fill the entire bottom. The tree would be a full, complete binary tree. It is full because every internal node has 2 children and all the leaves have 0 children. It is complete because the tree is filled with nodes to height -1 and all nodes at the bottom could be positioned on the left side (because there aren’t enough characters to fill the entire bottom of the Huffman tree). The only variable playing a major role in deciding the length of the code/codes is the number of characters or alphabet symbols. Frequency of the characters would not play a role because each character has an equal frequency. As a result of this conclusion and the fact that not all of the characters will be located at the very bottom of the tree, some characters will have one binary digit less than others because they are one level higher compared to the characters at the bottom of the Huffman Tree. So, the deciding factor in the length of the code is the number of characters. |

**Q2: [8 points]** What is the compression ratio if you use Huffman coding to compress the following term? hippopotomonstrosesquippedaliophobia

Notes:

* hippopotomonstrosesquippedaliophobia: it a s phobia where a person fears long words.
* Compression ratio=(total number of bits after compression)/(total number of bits before compression) \* 100
* No code is required.

|  |
| --- |
| Use this space to answer the question  Compression ratio = 132 / 288 = 0.4583 \* 100 = 45.83%  Compression ratio = 45.83% |

**Q3: [4 points]** If you were using a map to store the following lists of items, which data field would you select as the key and why?

a. textbook title, author, ISBN, year, publisher

b. player’s name, uniform number, team, position

c. computer manufacturer, model number, processor, memory, disk size

d. department, course title, course ID, section number, days, time, room

|  |
| --- |
| Use this space to answer the question   1. ISBN, because each book has a unique ISBN (no duplicates). 2. Uniform Number, because each team can only have one unique uniform number per player (no duplicates). 3. Model Number, because each computer has a unique model number (no duplicates). 4. Course ID, because each course has a unique course ID (no duplicates). |

**Q4: programming question [8 points]** Declare the map container stateDataMap to store pairs of the form (stateName, capitalName), where stateName and capitalName are variables of type string:

* Write C++ statements that add the following pairs to stateDataMap: (Nebraska, Lincoln), (New York, Albany), (Ohio, Columbus), (California, Sacramento), (Massachusetts, Boston), and (Texas, Austin).
* Write a C++ method that outputs the data stored in stateDataMap.
* Write a C++ statement that changes the capital of California to Los Angeles
* Write a C++ method that outputs the capitalName using the stateName which will be entered by the user.

|  |
| --- |
| Use this space to answer the question  //Chad Huntebrinker  //CS 303  //Problem 4  #include <iostream>  #include <map>  #include <string>  using namespace std;  void output\_data(map<string, string> myMap);  map<string, string> change\_Cali(map<string, string> myMap);  void output\_user\_state(map<string, string> myMap, string userInput);  int main() {  //Declare the map stateDataMap.  map<string, string> stateDataMap;  string userInput;  //Insert the states and state capitals.  stateDataMap.insert(pair<string, string>("Nebraska", "Lincoln"));  stateDataMap.insert(pair<string, string>("New York", "Albany"));  stateDataMap.insert(pair<string, string>("Ohio", "Columbus"));  stateDataMap.insert(pair<string, string>("California", "Sacramento"));  stateDataMap.insert(pair<string, string>("Massachusetts", "Boston"));  stateDataMap.insert(pair<string, string>("Texas", "Austin"));  //Function that outputs data.  output\_data(stateDataMap);  //Function that changes the value of California.  stateDataMap = change\_Cali(stateDataMap);  cout << "Enter the state to see it's capital." << endl;  cout << "States you can enter: " << endl;  cout << "Nebraska" << endl;  cout << "New York" << endl;  cout << "Ohio" << endl;  cout << "California" << endl;  cout << "Massachusetts" << endl;  cout << "Texas" << endl << endl;  getline(cin, userInput);  //Function that outputs the value of the key the user enters in.  output\_user\_state(stateDataMap, userInput);  return 0;  }  //This function outputs all the values and keys of the map.  void output\_data(map<string, string> myMap) {  cout << "Nebraska, " << myMap["Nebraska"] << endl;  cout << "New York, " << myMap["New York"] << endl;  cout << "Ohio, " << myMap["Ohio"] << endl;  cout << "California, " << myMap["California"] << endl;  cout << "Massachusetts, " << myMap["Boston"] << endl;  cout << "Texas, " << myMap["Texas"] << endl << endl;  }  //This function changes the value of the key California.  map<string, string> change\_Cali(map<string, string> myMap) {  myMap.at("California") = "Los Angeles";  return myMap;  }  //This function outputs the value of the key the user enters in,  //if it exists.  void output\_user\_state(map<string, string> myMap, string userInput) {  //If the user's key exists in the map.  if (myMap.find(userInput) != myMap.end()) {  cout << myMap.at(userInput) << endl << endl;  }  //Else if the user's key doesn't work.  else {  cout << userInput << " does not exist in the map." << endl << endl;  }  } |