|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EHB354E FINAL EXAM - 10/6/2023  ANSWERS FILE |  | |  |  | | --- | --- | | Student Number : | 040180063 | | Full Name : | Yiğit Bektaş Gürsoy | |

• Write your answers inside the given blank boxes below.

• For coding questions (Q1 and Q3) , use keyboard only.

**ANSWER 1)** [20 points]

|  |
| --- |
| #include <iostream>  #include <map>  #include <vector>  #include <fstream>  #include <algorithm>  using namespace std;  // Subcategory class  class Subcategory {  public:      string name;      int amount;      Subcategory(string n, int a) : name(n), amount(a) {}      void print() {          cout << "Name: " << name << ", Amount: " << amount << endl;      }  };  // Comparison function for sorting  bool compareSubcategory(const Subcategory &a, const Subcategory &b) {      return a.amount < b.amount;  }  int main() {      // Map      map<string, vector<Subcategory>> M;      // Initialize map      M["ELECTRONICS GROUP"] = {          Subcategory("Router", 41), Subcategory("Battery", 136), Subcategory("Monitor", 30),          Subcategory("Printer", 24), Subcategory("Audio", 43), Subcategory("Navigation", 15),          Subcategory("Video", 32), Subcategory("Modem", 26)      };      M["MACHINERY GROUP"] = {          Subcategory("Shipping", 23), Subcategory("Crane", 7), Subcategory("Hydraulics", 15),          Subcategory("Pneumatics", 10), Subcategory("Forklift", 8), Subcategory("Carrier", 13)      };      M["STATIONERY GROUP"] = {          Subcategory("Paper Clip", 1370), Subcategory("Envelope", 560), Subcategory("Ink", 140),          Subcategory("Notepad", 710), Subcategory("Paper Pack", 230), Subcategory("Clipboard", 1720),          Subcategory("Binder", 690)      };      ofstream file("Output.html");      // Loop over map      for (auto &group : M) {          // Sort subcategories by amount          sort(group.second.begin(), group.second.end(), compareSubcategory);          int totalAmount = 0;          file << "<table border=1>\n";          file << "<tr>\n<td align=center colspan=2 bgcolor=aqua >" << group.first << "</td>\n</tr>\n";          file << "<tr>\n<td> SUBCATEGORY NAME </td>\n<td> AMOUNT </td>\n</tr>\n";          // Loop over subcategories          for (auto &subCategory : group.second) {              totalAmount += subCategory.amount;              file << "<tr>\n<td> " << subCategory.name << " </td>\n<td align=center> " << subCategory.amount << " </td>\n</tr>\n";          }            file << "<tr>\n<td> TOTAL AMOUNT </td>\n<td align=center> " << totalAmount << " </td>\n</tr>\n";          file << "</table>\n<br>\n";      }      file.close();      return 0;  } |

**ANSWER 2)** [10 points]

(Draw with Word drawing tools, or add picture of hand-drawn answer on paper.)

|  |
| --- |
|  |

**ANSWER 3)** [70 points]

|  |
| --- |
| #include <iostream>  #include <list>  #include <string>  #include <algorithm>  #include <cctype>  #include <ctime>  using namespace std;  struct Employee {   int ID;   string Name;  };  class BST {  public:      struct Node {          Employee emp;          Node \*left, \*right;      };      string deptname;      Node \* root;      BST (string deptname, Employee kaynak[] , int N) : root(nullptr), deptname(deptname) {          for(int i=0; i<N; i++) {              ADD(kaynak[i]);          }      }      bool ADD (Employee yeni) {          Node\* newNode = new Node{yeni, nullptr, nullptr};          if(root == nullptr){              root = newNode;          } else {              Node\* current = root;              while(current){                  if(yeni.ID == current->emp.ID) return false;                  if(yeni.ID < current->emp.ID){                      if(current->left == nullptr){                          current->left = newNode;                          return true;                      } else {                          current = current->left;                      }                  } else {                      if(current->right == nullptr){                          current->right = newNode;                          return true;                      } else {                          current = current->right;                      }                  }              }          }          return true;      }      BST (int N) : root(nullptr), deptname("Testing") {          srand(time(0));          for(int i=0; i<N; i++) {              Employee emp;              emp.ID = rand();              emp.Name = to\_string(rand());              ADD(emp);          }      }      BST (const BST & other\_tree) : root(nullptr), deptname(other\_tree.deptname) {          DUPLICATE(other\_tree.root);      }      void DUPLICATE (Node \* otherP) {          if(otherP) {              ADD(otherP->emp);              DUPLICATE(otherP->left);              DUPLICATE(otherP->right);          }      }      Employee \* SEARCH (string isim) {          Node \* current = root;          transform(isim.begin(), isim.end(), isim.begin(), ::toupper);          while(current){              string currentName = current->emp.Name;              transform(currentName.begin(), currentName.end(), currentName.begin(), ::toupper);              if(currentName.find(isim) != string::npos) {                  return &(current->emp);              } else if(isim < currentName){                  current = current->left;              } else {                  current = current->right;              }          }          return nullptr;      }      Employee \* SEARCH (int idnum, Node \* P) {          if(P){              if(P->emp.ID == idnum) return &(P->emp);              if(idnum < P->emp.ID) return SEARCH(idnum, P->left);              return SEARCH(idnum, P->right);          }          return nullptr;      }      void DISPLAY (Node \* P) {          if(P) {              DISPLAY(P->left);              cout << "ID: " << P->emp.ID << ", Name: " << P->emp.Name << endl;              DISPLAY(P->right);          }      }  };  int main() {      list<BST> trees;      Employee distEmployees[] = {{101, "Sunay"}, {102, "Cihan"}, {103, "Üner"}, {104, "Kaptan"}};      BST dist("Distribution", distEmployees, 4);      trees.push\_back(dist);      Employee accEmployees[] = {{201, "Şenol"}, {202, "Aktaş"}, {203, "Kaplan"}, {204, "Sungur"}};      BST acc("Accounting", accEmployees, 4);      trees.push\_back(acc);      Employee prodEmployees[] = {{301, "Çakır"}, {302, "Sunay"}, {303, "Acar"}};      BST prod("Production", prodEmployees, 3);      trees.push\_back(prod);      Employee markEmployees[] = {{401, "İlker"}, {402, "Vedat"}, {403, "Ceyhan"}};      BST mark("Marketing", markEmployees, 3);      trees.push\_back(mark);      BST distCopy(dist);      trees.push\_back(distCopy);      BST randomTree(6);      trees.push\_back(randomTree);      for(auto& tree : trees) {          cout << "Department: " << tree.deptname << endl;          tree.DISPLAY(tree.root);          cout << endl;      }      while(true){          string input;          cout << "Enter a search value (Employee ID integer or Employee Name string): ";          cin >> input;          if(isdigit(input[0])){              int id = stoi(input);              for(auto& tree : trees) {                  Employee\* emp = tree.SEARCH(id, tree.root);                  if(emp){                      cout << "Found in " << tree.deptname << ", ID: " << emp->ID << ", Name: " << emp->Name << endl;                  }              }          } else {              for(auto& tree : trees) {                  Employee\* emp = tree.SEARCH(input);                  if(emp){                      cout << "Found in " << tree.deptname << ", ID: " << emp->ID << ", Name: " << emp->Name << endl;                  }              }          }      }      return 0;  } |