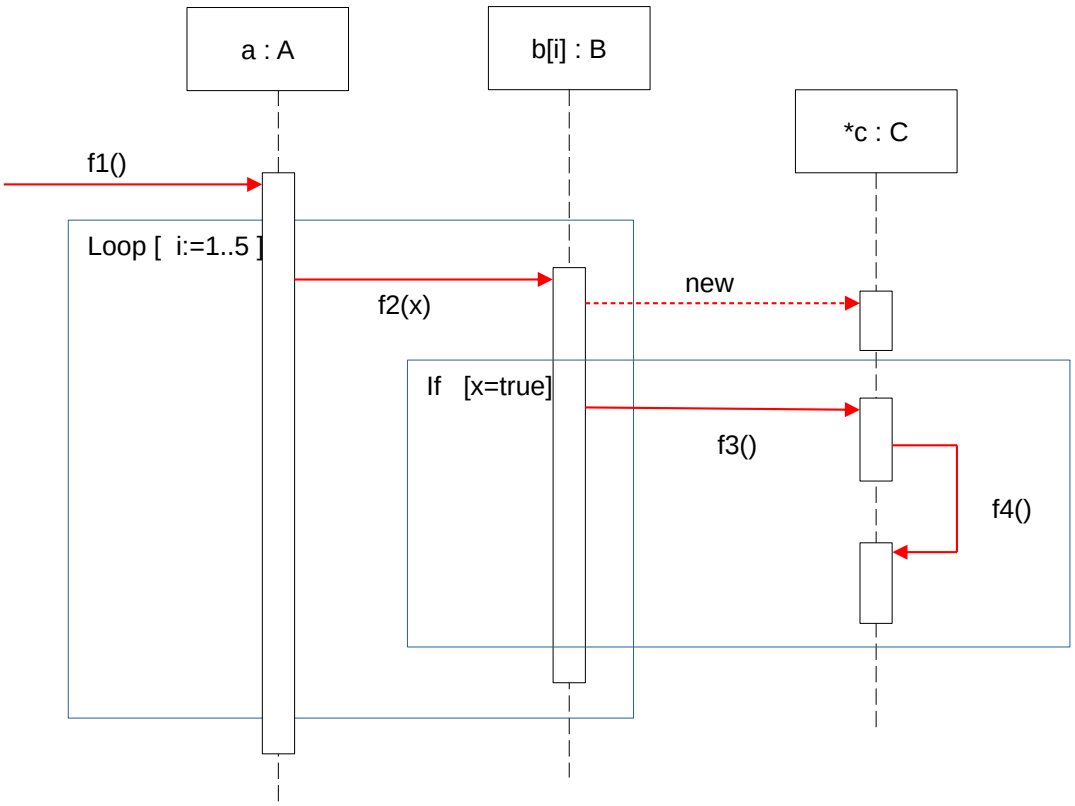
ANSWER 1) [20 points]

```
#include <map>
#include <vector>
#include <algorithm>
#include <fstream>

using namespace std;

ofstream  dosya ("output.html", ios::out); //Global

class Subcategory
{
  public:
  string name;
  int amount;
  Subcategory(string n, int a) : name(n), amount(a) {}
  void print() {
     dosya << "<tr>" << endl;
     dosya << "<td>";

     dosya << name;

     dosya << "</td>" << endl;
     dosya << "<td align=center>";

     dosya << amount;

     dosya << "</td>" << endl;
     dosya << "</tr>" << endl;
     dosya << endl << endl;
  }
};


bool karsilastir(Subcategory s1, Subcategory s2)
{
  return s1.amount < s2.amount ? true : false;
}

int main()
{
 int Total=0;

 map < string, vector <Subcategory> >  M;
 M["ELECTRONICS"] = { {"Router",41} , {"Battery",136} , {"Monitor",30} , {"Printer",24} , {"Audio",43} ,
                {"Navigation",15} , {"Video",32} , {"Modem",26} };
 M["MACHINERY"]   = { {"Shipping",23} , {"Crane",7} , {"Hydraulics",15} , {"Pneumatics",10} , {"Forklift",8} , {"Carrier",13} };
 M["STATIONERY"]  = { {"Paper Clip",1370} , {"Envelope",560} , {"Ink",140} , {"Notepad",710} ,
                {"Paper Pack",230} , {"Clipboard",1720} , {"Binder",690} };

 for (auto harita : M)
  {
      dosya << "<table border=1>" << endl << endl;
      dosya << "<tr>" << endl;
      dosya << "<td align=center colspan=2 bgcolor=aqua >";

      dosya << harita.first; //Group name

      dosya << " GROUP </td>" << endl;
      dosya << "</tr>" << endl << endl;

      dosya << "<tr>" << endl;
      dosya << "<td> SUBCATEGORY NAME </td>" << endl;
      dosya << "<td> AMOUNT </td>" << endl;
      dosya << "</tr>" << endl << endl;

      Total = 0;
```

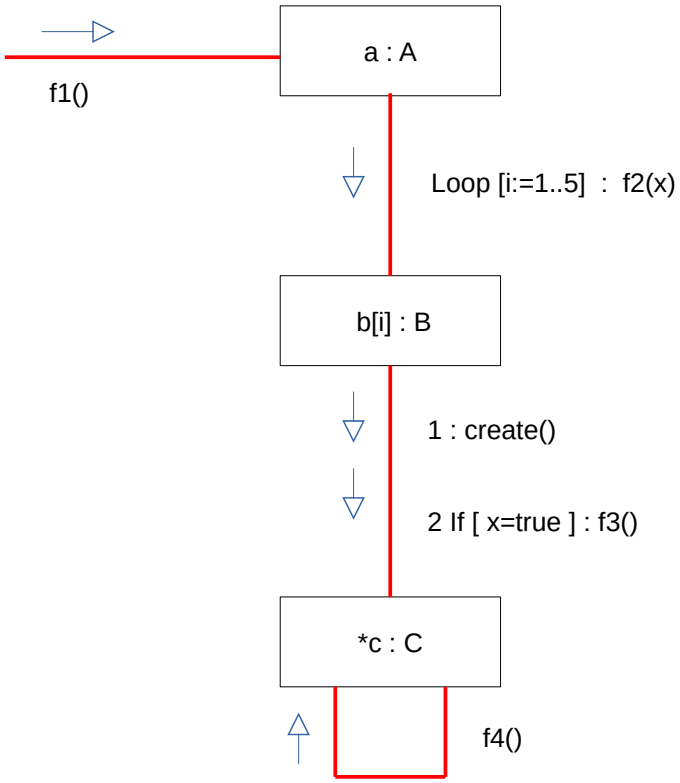
```
    sort(harita.second.begin(), harita.second.end(), karsilastir);

    for (auto i : harita.second)
    {
       i.print(); //Subcategory name and amount
       Total += i.amount;
    }

   dosya << "<tr>" << endl;
   dosya << "<td>TOTAL AMOUNT";
   dosya << "</td>" << endl;
   dosya << "<td align=center>";

   dosya << Total;

   dosya << "</td>" << endl;
   dosya << "</tr>" << endl;
   dosya << endl << endl;

   dosya << "</table>" << endl << endl;
   dosya << "<br>" << endl << endl;
 }

  dosya.close();
} //End of main
```

ANSWER 2) [10 points]

UML Sequence Diagram



UML Communication Diagram



ANSWER 3) [70 points]

```
#include <iostream>
#include <list>
#include <ctime>   // time function
#include <cstdlib> // srand and rand functions
#include <cstring> // strstr function
#include <locale>
using namespace std;

//---------------------------------------------------------------------
struct Employee {   // Employee struct definition
  int ID;        // Employee ID number (unique)
  string Name;    // Employee surname (not-unique)
};

//---------------------------------------------------------------------
// Binary Search Tree
class BST
{
public:
  struct Node {        // Nested struct definition
    Employee  emp;      // Data
    Node *left, *right;  // Pointers to left and right branches
  };
 string deptname;                    // Department name
 Node * root;                       // Pointer to top node
 BST (string deptname, Employee kaynak[] , int N); // Parametered constructor1
 bool ADD (Employee  yeni);              // Iterative
 BST (int N);                       // Parametered constructor2
 BST (const  BST &  other_tree);        // Copy constructor
 void DUPLICATE (Node * otherP);         // Recursive
 Employee * SEARCH (string isim);        // Iterative
 Employee * SEARCH (int idnum, Node * P);  // Recursive (overloaded)
 void DISPLAY (Node * P);               // Recursive
}; // end of class
```

```
//------------------------------------------------------------------------
// Constructor1
 BST :: BST (string dn, Employee kaynak[] , int N) : deptname(dn)
 {
 root = NULL;
 for (int i=0; i < N; i++)
    ADD( kaynak[i] );
 } // end of constructor1

//------------------------------------------------------------------------
// Constructor2
 BST :: BST ( int N )
 {
 root = NULL;
 deptname = "Testing";
 srand(time(NULL));

 Employee yeni;

 for (int i=1; i <= N; i++)
 {
    yeni.ID = rand()%1000 + 1000;
    char harf = 'A' + rand()%26;
    yeni.Name = { harf, harf, harf, harf, harf, harf };
    ADD(yeni);
 }
 } // end of constructor2

//------------------------------------------------------------------------
// Copy constructor
 BST :: BST ( const  BST &other_tree)
 {
  this->deptname = other_tree.deptname;
  this->root = NULL;
  DUPLICATE (other_tree.root);
 } // end of constructor

//------------------------------------------------------------------------
// Iterative
 bool BST :: ADD (Employee yeni)
 {
 Node * cur;  // Current
 Node * prev; // Previous
 Node * n;   // New node

 n = new Node;
 n->emp  = yeni;  //Employee
 n->left = n->right = NULL;

 if (root == NULL) { // Check if tree is empty
   root = n;       // First node is added
   return false;
 }

 // Find a suitable leaf node for adding location.
 cur = root; // Start looping from root
 while (cur != NULL) {
  if (yeni.ID == cur->emp.ID)  //Check for uniqueness
     return false; //Stop looping, Add failed

  prev = cur;
  if (yeni.Name < cur->emp.Name)
     cur = cur->left;  // Go to left branch
  else
     cur = cur->right; // Go to right branch
 }

 // A leaf node has been reached, add the new node to leaf.
 if (yeni.Name < prev->emp.Name)
```

```
    prev->left = n;
  else
    prev->right = n;

  return true;
} // end of function

//-------------------------------------------------------------------------
// Recursive
  void BST :: DUPLICATE (Node * otherP)
  {
   if ( otherP != NULL )
   {
     ADD (otherP->emp);
     DUPLICATE (otherP->left );  //Go to left branch
     DUPLICATE (otherP->right ); //Go to right branch
   }
  } // end of function

//-------------------------------------------------------------------------
string buyukharf(string s) // Converts a string to all uppercase
{
              for (int i=0; i < s.length(); i++)
                 s[i] = toupper(s[i]);
              return s;
}

//-------------------------------------------------------------------------
// Iterative
  Employee * BST :: SEARCH (string isim)
 {
  Node * cur;  // Current
  cur = root;  // Start searching from root


  string uisim = buyukharf( isim );

  string uname;
  // Search until a leaf node is reached.
  // //Using built-in string functions
  while (cur != NULL)
  {
   uname = buyukharf( cur->emp.Name );

   if ( strstr( uname.c_str(), uisim.c_str() ) != NULL ||
      strstr( uisim.c_str(), uname.c_str() ) != NULL ) //Match found
                break;

   if (uisim < uname)
     cur = cur->left; // Go to left branch
   else
     cur = cur->right; //Go to right branch
  } // end of while

  if (cur != NULL)
    return &(cur->emp); //Match found, return pointer to Employee struct
  else
    return NULL; //Not found

 } // end of function

//-------------------------------------------------------------------------
  // Overloaded function (Recursive)
  Employee * BST :: SEARCH (int idnum, BST :: Node  * P)
 {
   if ( P != NULL )
   {
     if (P->emp.ID == idnum)
                return &(P->emp); // Exact match found, return pointer to Employee struct
```

```
    Employee *q = NULL;

    q = SEARCH(idnum, P->left ); //Go to left branch
    if (q != NULL)
      return q; // Exact match found

    q = SEARCH(idnum, P->right ); //Go to right branch
    if (q != NULL)
      return q; // Exact match found
  }

  return NULL; // Not found
}

//-------------------------------------------------------------------
//Recursive
 void BST :: DISPLAY (BST :: Node  * P)
 {
  if ( P != NULL )
  {
    DISPLAY( P->left ); //Go to left branch
    cout << P->emp.ID << " - " << P->emp.Name << endl; //Employee info
    DISPLAY( P->right ); //Go to right branch
  }
 } // end of function

//*************************************************************************
int main()
{
 setlocale(LC_ALL, ""); //Türkçe

 Employee E1[4] = { {101,"Sunay"} , {102,"Cihan"} , {103, "Üner"} , {104, "Kaptan"} };
 Employee E2[4] = { {201,"Şenol"} , {202,"Aktaş"} , {203,"Kaplan"} , {204,"Sungur"} };
 Employee E3[3] = { {301,"Çakır"} , {302,"Sunay"} , {303,"Acar"}  };
 Employee E4[3] = { {401,"İlker"} , {402,"Vedat"} , {403,"Ceyhan"} };

 list <BST> agaclar; //STL list of trees
 agaclar.push_back ( BST("Distribution", E1, 4) ); // Tree1 : Parametered constructor1
 agaclar.push_back ( BST(6) );         // Tree2 : Parametered constructor2 (generates random data)
 agaclar.push_back ( *agaclar.begin() );          // Tree3 : Copy constructor of first tree in STL list
 agaclar.push_back ( BST("Accounting", E2, 4) );   // Tree4 : Parametered constructor1
 agaclar.push_back ( BST("Production", E3, 3) );   // Tree5 : Parametered constructor1
 agaclar.push_back ( BST("Marketing", E4, 3) );    // Tree6 : Parametered constructor1

 //-------------------------------------------------------
 int sayac = 1;
 for (auto agac : agaclar)
 {
   cout << "TREE" << sayac << " : \n";
            cout << "Department : " << agac.deptname << endl;
            agac.DISPLAY (agac.root);
   cout << "----------------------------\n";
   sayac++;
 }

 //-------------------------------------------------------
 Employee * sonuc;
 string aranan;
while (1) //Endless loop
{
 cout << "\nSearch in all departments (Ctrl+C exits from program) \n";

 cout << "Enter a search value (Employee ID integer or Employee Name string) : ";
 cin >> aranan;

 sayac = 0;
 list <BST> ::  iterator  j;
```

```
 for (j = agaclar.begin(); j != agaclar.end(); j++)
 {
   if ( isdigit( aranan[0] ) )  // Checking first character for integer
   {
     int ara_num = atoi(aranan.c_str() );  // Convert string to const char*, then to integer
     sonuc = j->SEARCH(ara_num, j->root); //Call recursive function
   }
   else
     sonuc = j->SEARCH(aranan); //Call iterative function

   if (sonuc != NULL)
   {
     sayac++;
     cout << "\nFound in Tree" << sayac << " : " << endl;
               cout << "Department : " << j->deptname;
     cout << sonuc->ID << " - " << sonuc->Name << endl; //Employee
   }
 }

 cout << "\n(" << sayac << ") RESULTS HAVE BEEN FOUND\n\n";
} // Endless loop

} // end of main
```