A. The College Member Class

CollegeMember
protected int mID_Number protected String mFirstName, mLastName, mGender protected int mDateOfBirth protected String mTelephone, mEmail public static final int DEFAULT_ID = 0
public CollegeMember() public CollegeMember(CollegeMember thisMember) public void modifyMe(CollegeMember thisMember) public String printMe() public void inputData(int x, String inCategory) public String toString() public int getID() protected void finalize() public boolean equals(Object thisObject) public int compareTo(Object thisObject)

A1. The CollegeMember() Constructor

```
START
    mID_Number, mDateOfBirth := DEFAULT_ID;
    mFirstName, mLastName, mGender, mTelephone, mEmail := " ";
STOP
```

A2. The CollegeMember(CollegeMember thisMember) Overloaded Constructor

```
START
    mID_Number := thisMember.mID_Number;
    mLastName := thisMember.mLastName;
    mFirstName := thisMember.mFirstName;
    mGender := thisMember.mGender;
    mDateOfBirth := thisMember.mDateOfBirth;
    mEmail := thisMember.mEmail;
    mTelephone := thisMember.mTelephone;
STOP
```

A3. The void modifyMe(CollegeMember thisMember) Method

```
START
    mID_Number := thisMember.mID_Number;
    mLastName := thisMember.mLastName;
    mFirstName := thisMember.mFirstName;
    mGender := thisMember.mGender;
    mDateOfBirth := thisMember.mDateOfBirth;
    mEmail := thisMember.mEmail;
    mTelephone := thisMember.mTelephone;
STOP
```

A4. The int getID() Method

START

Return mID_Number;

STOP

A5. The void inputData(int x, String inCategory) Method

START

Let inputID, inputTele, inputFName, inputLName, inputDoB be strings;
Prompt for and accept inputID for inCategory x;
While (NOT validateID(inputID)) do the following:
DisplayMessage("ID Number must be numeric");
Prompt for and accept inputID for inCategory x;
End-While;
Prompt for and accept inputFName for inCategory x;
While (inputName' s first character is not a letter) do the following:
DisplayMessage("Name must begin with a letter");
Prompt for and accept inputFName for inCategory x;
End-While;
Prompt for and accept inputLName for inCategory x;
While (inputName' s first character is not a letter) do the following:
DisplayMessage("Name must begin with a letter");
Prompt for and accept inputLName for inCategory x;
End-While;
Prompt for and accept mEmail for inCategory x
Prompt for and accept mGender for inCategory x;
Prompt for and accept inputTele for inCategory x;
While (NOT validateTele(inputTele)) do the following:
DisplayMessage("Telephone number is not in the required format");
Prompt for and accept inputTele for inCategory x;
End-While;
Prompt for and accept inputDoB for inCategory x;
While (NOT validateDoB(inputDoB)) do the following:
DisplayMessage("Invalid Date of Birth");
Prompt for and accept inputDoB for inCategory x;
End-While;
mID_Number := inputID;
mFirstName := inputFName;
mLastName := inputLName;
mTelephone := inputTele;
mDateOfBirth := inputDoB;

STOP

A6. The boolean validateTele(String thisTele) Method

START

Let isValid be Boolean, initialized to true;
Let x be an integer;

```
        For (x:= 1 to 12 with increments of 1) do the following:
                Case x is
                4, 8: If (thisTele.CharacterAt(x-1) <> '-') isValid := false; EndIf;
                Otherwise: If (thisTele.CharacterAt(x-1) is not a digit) isValid := false; EndIf;
                End-Case;
        End-For
        Return isValid;
STOP
        A7. The boolean validateID(String thisID) Method
START
        Let isValid be Boolean, initialized to true;
        Let x be an integer;
        For (x:= 1 to thisID.length() with increments of 1) do the following:
                If (thisID.CharacterAt(x-1) is not a digit) isValid := false; EndIf;
        End-For;
        Return isValid;
STOP
        A8. The boolean validateDoB(String thisDate) Method
START
        Let isValid be Boolean, initialized to True;
        Let x, Year, Month, Day be integers;
        Let mCheck be an array of 13 integers;
        Let LeapYear be a Boolean flag;
        Let CurrentYear be the current year as retrieved from the system;
        Set Year to Substring(thisDate, 0,4);
        Set Month to Substring(thisDate,4,2);
        Set Day to Substring(thisDate, 6,2);
        LeapYear:= False;
        If (Year mod 400) is 0) OR ((Year Mod 4) is 0 AND (Year Mod 100) <> 0)) LeapYear :=
True; End-If;
        mCheck[0] := 0; mCheck[1] := 31; mCheck[2] := 28; If (LeapYear ) mCheck[2] := 29;
End-If
        mCheck[3], mCheck[5], mCheck[7], mCheck[8]; mCheck[10], mCheck[12] := 31;
        mcheck[4], mCheck[6], mCheck[9], mCheck[11] := 30;
        If (Year > CurrentYear) isValid := False; End-If;
        Else If (Month < 1 OR Month > 12) isValid := False; End-If;
                Else If (Day > mCheck[Month]) isValid := False; End-If;
        Return isValid;
STOP
        A9. The String printMe( ) Method
START
        Let printString be a string;
```

printString := "ID Number: " + mID_Number +" Name: " + mFirstName + " "+ mLastName + + "Gender: " + mGender + ", " + " Date of Birth: " + mDateOfBirth + + "mTelephone: " + mTelephone + + "E-Mail: " + mEmail;
Return printString;
STOP

A10. The void finalize() Method
START
Destroy the current object and call the garbage collection routine;
STOP

A11. The String toString() Method
START
Return "CollegeMember: " + this.printMe();
STOP

A12. The boolean equals(Object thisObject) Method
START
Let instanceMatch be Boolean, initialized to false;
If (thisObject is an instance of CollegeMember)
    Let thisMember be a CollegeMember object, instantiated by thisObject;
    If (mID_Number = thisMember.mID_Number) instanceMatch := true; End-If;
End-If;
Return instanceMatch;
STOP

B. The Generic Class

Generic
protected String alumAcadDept = Null, alumMajor= Null protected String alumCurrentEmployer= Null, alumJobTitle= Null protected String stdAcadDept= Null, stdAcadMajor= Null protected String empJobDept= Null, empJobSpecialization= Null, empJobTitle= Null Protected int setDataType Protected String dataTypeString = “”
public Generic(CollegeMember thisMember) public void modifyMe(Generic thisItem) public void inputData(int x) public String printMe() public String toString() protected void finalize() public boolean equals(Object thisObject) public void setDataType(int dataType)

B1. The Generic(CollegeMember thisMember) Constructor
START
super (thisMember);

```
        //based on which data Item is being chosen
        Case setDataType is
                1: empJobDept, empJobSpecialization, empJobTitle:= " ";
                2:stdAcadDept, stdAcadMajor:= " ";
                3:alumAcadDept , alumMajor, alumCurrentEmployer, alumJobTitle:= " ";
        End-Case;
STOP
```

B2. The void modifyMe(Generic thisItem) Method

```
START
        Let x be an integer;
        super.modifyMe(thisItem);
        Case setDataType is
                1:empJobDept := thisItem.empJobDept; empJobSpecialization:=
        thisEmp.empJobSpecialization; empJobTitle:= thisEmp.empJobTitle;
                2:stdAcadDept := thisItem.stdAcadDept; stdAcadMajor:= thisItem.stdAcadMajor;
                3:alumAcadDept  := thisAlum.alumAcadDept ; alumMajor:= thisAlum.alumMajor;
alumCurrentEmployer := thisAlum.alumCurrentEmployer; alumJobTitle:= thisAlum.alumJobTitle;
        End-Case;
STOP
```

B3. The public void inputData(int x) Method

```
START
        super.inputData(x, dataTypeString);
        Case setDataType is:
                1:Prompt for and accept empJobDept for employee x; Prompt for and accept
empJobSpecialization for employee x; Prompt for and accept empJobTitle for employee x;
                2:Prompt for and accept stdAcadDept for student x; Prompt for and accept
stdAcadMajor for student x;
                3:Prompt for and accept alumAcadDept for alumnus x; Prompt for and accept
alumMajor for alumnus x; Prompt for and accept alumCurrentEmployer  for alumnus x; Prompt
for and accept alumJobTitle for alumnus x;
STOP
```

B4. The public String printMe() Method

```
START
        Let printString be a string;
        Let x be an integer;
        Case setDataType is:
                1:printString := super.printMe() + + "Department: " + Department + +
"Specialization: " + Specialization + + "Job Title: " + jobTitle;
                2:printString := super.printMe() + + "Department: " + acadDept + + "Major: " +
acadMajor;
```

3:printString := super.printMe() + + "Department: " + acadDept + + "Major: " + acadMajor; + + “Current Employer “ + currentEmployer + + “Job Title: “ + jobTitle;
Return printString;
STOP

B5. The public String toString() Method
START
Return dataTypeString + this.printMe();
STOP

B6. The protected void finalize() Method
START
Destroy the current object and call the garbage collection routine;
STOP

B7. The public boolean equals(Object thisObject) Method
START
Let instanceMatch be Boolean, initialized to false;
If (thisObject is an instance of Generic)
Let thisMember be a Generic object, instantiated by thisObject;
If (mID_Number = thisMember.mID_Number) instanceMatch := true; End-If;
End-If;
Return instanceMatch;
STOP

B8. The void setDataType(int dataType) Method
START
setDataType = dataType
Case dataType is:
1:dataTypeString = “Employee”
2:dataTypeString = “Student”
3:dataTypeString = “Alumnus”
STOP

C. The GenericNode Class

GenericNode
Protected Generic nInfo Protected GenericNode nLeft, nRight
GenericNode() Void modifyMe(GenericNode thisNode) Void inputData(int x) String printMe

C1. The GenericNode Constructor()

START

Instantiate nInfo based on Generic;
nLeft, nRight := NULL;

STOP

C2. The void modifyMe(GenericNode thisNode) Method

START

nInfo.modifyMe(thisNode.nInfo);
nLeft := thisNode.nLeft;
nRight := thisNode.nRight;

STOP

C3. The void inputData(int x) Method

START

nInfo.inputData(x);
nLeft, nRight := NULL;

STOP

C4. The String printMe() Method

START

Return nInfo.printMe();

STOP

D. The GenericBinaryTree Class

GenericBinaryTree
Protected GenericNode root protected int size, travRef ArrayList <Generic> travResult
public GenericBinaryTree() public void addRoot(Generic thisStud) public void addLeftLeaf(GenericNode thisLeaf, Generic thisItem) public void addRightLeaf(GenericNode thisLeaf, Generic thisItem) leaf public void addLeftSubtree(GenericNode thisNode, GenericNode newNode) public void addRightSubtree(GenericNode thisNode, GenericNode newNode) Void insert(GenericNode thisNode, Generic thisItem) StudentNode findInsertionPoint(GenericNode thisNode, Generic thisItem) public void removeSubtree(GenericNode thisNode) public StudentNode Search(Generic searchArg) public void modifyMe(GenericNode thisNode, Generic thisItem) public void clearTree() public Student getInfo(GenericNode thisNode) public StudentNode getNode(GenericNode thisNode) public int getSize() public void setSize(GenericNode thisNode)

```
public boolean isEmpty()
public void inOrderPrep()
public void inOrderTraversal(GenericNode thisNode)
```

```
        D1.The GenericBinaryTree() Constructor
START
        root := NULL;
        size, travRef := 0;
        travResult := NULL;
STOP
        D2.The void addRoot(Generic thisItem) Method
START
        If (root = NULL)
                Instantiate root;
                root.nInfo.modifyMe(thisItem);
                Set root.nLeft and root.nRight to NULL;
                Add 1 to size;
        End-If;
STOP
        D3.The void addLeftLeaf(GenericNode thisLeaf, Generic thisItem) Method
START
        Let newNode be a GenericNode;
        Instantiate newNode;
        newNode.nInfo.modifyMe(thisItem);
        Set newNode.nLeft and newNode.nRight to NULL;
         thisLeaf.nLeft := newNode;
        Add 1 to size;
STOP
        D4.The void addRightLeaf(GenericNode thisLeaf, Generic thisItem) Method
START
        Let newNode be a GenericNode;
        Instantiate newNode;
        newNode.nInfo.modifyMe(thisItem);
        Set newNode.nLeft and newNode.nRight to NULL;
        thisLeaf.nRight := newNode;
        Add 1 to size;
STOP
        D5.The void addLeftSubtree(GenericNode thisNode, GenericNode newNode) Method
START
        Let Temp, leftEnd be GenericNode references;
        Temp := thisNode.nLeft;
        leftEnd := newNode;
        While (leftEnd.nLeft <> NULL) leftEnd := leftEnd.nLeft; End-While;
        thisNode.nLeft := newNode;
```

```
        leftEnd.nLeft := Temp;
        Increase size by size of the sub-tree
STOP
        D6.The void addRightSubtree(GenericNode thisNode, GenericNode newNode) Method
START
        Let Temp, rightEnd be StudentNode references;
        Temp := thisNode.nRight;
        rightEnd := newNode;
        While (rightEnd.nRight <> NULL) rightEnd := rightEnd.nRight; End-While;
        thisNode.nRight := newNode;
        rightEnd.nRight := Temp;
        Increase size by the size of the sub-tree
STOP
        D7.The void insert (GenericNode thisNode, Generic thisitem) Method
START
        Let newNode and currentNode be instances of StudentNode;
        If (thisNode = NULL)
                Instantiate newNode;
                newNode.nInfo.modifyMe(thisItem);
                Set newNode.nLeft and newNode.nRight to NULL;
                root := newNode;
        Else
                currentNode := findInsertionPoint(thisNode, thisItem);
                 If (thisItem.getID() < currentNode.nInfo.getID())
                        addLeftLeaf(currentNode, thisItem);
                Else addRightLeaf(currentNode, thisItem);
                End-If;
        End-If;
        Add 1 to size;
STOP
        D8.GenericNode findInsertionPoint(GenericNode thisNode, Generic thisItem) Method
START
        Let insertPoint be a GenericNode;
        insertPoint := thisNode;
        If (thisitem.getID() < insertPoint.nInfo.getID())
                If (insertPoint.nLeft <> NULL)
                        insertPoint := findInsertionPoint(insertPoint.nLeft, thisItem);
                End-If;
        Else // thisItem >= insertPoint.nInfo
                If (insertPoint.nRight <> NULL)
                        insertPoint := findInsertionPoint(insertPoint.nRight, thisItem);
                End-if;
        End-If;
        Return insertPoint;
```

```
STOP
        D9.The void removeSubtree(GenericNode thisNode)
START
        If (thisNode.nLeft = thisNode.nRight = NULL) // a leaf
                Kill(thisNode);
                Subtract 1 from size;
        Else
                If (thisNode.nLeft <> NULL) removeSubtree(thisNode.nLeft); End-If;
                If (thisNode.nRight <> NULL) removeSubtree(thisNode.nRight); End-If;
                Kill(thisNode);
        End-If;
STOP
        D10.The GenericNode directSearch(Generic searchArg) Method
START
        Let currentNode, soughtNode be GenericNode instances;
        currentNode := root; soughtNode = NULL;
        While (currentNode.nInfo.getID() <> searchArg.getID()) AND (currentNode.nInfo <>
NULL)
                If (searchArg.getID() < currentNode.nInfo. getID())
                        currentNode := currentNode.nLeft;
                Else
                        currentNode := currentNode.nRight;
                End-If;
        End-While;
        If (currentNode.nInfo.getID() = searchArg.getID()) // If item found
                // Instantiate soughtNode; soughtNode.modifyMe(currentNode);
                soughtNode := currentNode;
        End-If;
        Return soughtNode;
STOP
        D11.The void modifyMe(GenericNode thisNode, Generic thisItem) Method
START
        thisNode.nInfo.modifyMe(thisItem);
STOP
        D12.The void clearTree() Method
START
        removeSubtree(root);
STOP
        D13.The Generic getInfo(GenericNode thisNode) Method
START
        Return thisNode.nInfo;
STOP
        D14.The GenericNode getNode(GenericNode thisNode) Method
START
```

```
        Return thisNode;
STOP
        D15.The int getSize() Method
START
        setSize(root);
        Return size;
STOP
        D16.The boolean isEmpty() Method
START
        Return whether size is 0 or not;
STOP
        D17.The void inOrderPrep( ) Method
START
        Instantiate travResult to an empty String;
        setSize(root)
STOP
        D18.The void inOrderTraversal(GenericNode thisItem) Method
START
        Let anyItem be a Genericinstance;
        If (thisNode is not NULL)
                inOrderTraversal(thisNode.nLeft);
                Instantiate anyItem; anyItem.modifyMe(thisNode.nInfo);
                Append anyStud to travResult;
                inOrderTraversal(thisNode.nRight);
        End-If;
STOP
        D19.The void setSize (GenericNode thisNode) Method
START
        Let currentNode be a GenericNode instance;
        currentNode := thisNode;
        If (currentNode <> NULL)
                If (currentNode = root) Add 1 to size; End-If;
                If (currentNode.nLeft <> NULL)
                        Add 1 to size;
                        currentNode := currentNode.nLeft;
                        setSize(currentNode);
                End-If;
                If (currentNode.nRight <> NULL)
                        Add 1 to size;
                        currentNode := currentNode.nRight;
                        currentNode := currentNode.nRight;
                End-If;
        End-If;
STOP
```

E. The ForestMonitor Class

ForestMonitor
static GenericBinaryTree employeeTree static GenericBinaryTree studentTree static GenericBinaryTree alumnusTree final static String HEADING = "The College Forest of Petr Bowles"
public static void main(String[] args) private static void initialize() public static void inputStudents() public static void inputAlumni() public static void inputEmployees() public static void queryStudents() public static void queryAlumni() public static void queryEmployees() public static void removeStudents() public static void removeAlumni() public static void removeEmployees() public static void modifyStudents() public static void modifyAlumni() public static void modifyEmployees() public static void traverseStudents() public static void traverseAlumni() public static void traverseEmployees() public static void checkTreeSize() public static void emptyTree()

E1.The main(String[] args) Method

START

Set employeeTree.dataType to 1
Set studentTree.dataType to 2
Set alumnusTree.dataType to 3
Let exitTime be Boolean, initialized to False;
Let Option be an integer;
initialize();
While (Not exitTime) do the following:
Present the user with the following menu:
1. Enter Students Info
2. Enter Alumni Info
3. Enter Employees Info
4. Query Students Info
5. Query Alumni Info
6. Query Employees Info
7. Remove Students Info

8. Remove Alumni Info
9. Remove Employees Info
10.Modify Students Info
11.Modify Alumni Info
12.Modify Employees Info
13.Traverse Students Tree
14.Traverse Alumni Tree
15.Traverse Employees Tree
16.Check Tree Size
17.Empty Tree
90.Exit

Prompt user to key in the a menu selection and store this in Option;
Case Option is:
1: inputStudents();
2: inputAlumni();
3: inputEmployees();
4: queryStudents();
5: queryAlumni();
6: queryEmployees();
7: removeStudents();
8: removeAlumni();
9: removeEmployees();
10: modifyStudents();
11: modifyAlumni();
12: modifyEmployees();
13: traverseStudents();
14: traverseAlumni();
15: traverseEmployees();
16: checkTreeSize();
17: emptyTree();
90: Set exitTime to True;
End-Case;
End-While
STOP

E2.The void inputStudents ( ) Method

START
Let x, Limit be integers;
Let currentStud be an instance of Generic;
currentStud.setDataType(2);
Let dummyC be an instance of CollegeMember;
Instantiate dummyC;
Instantiate currentStud using dummyC as an intermediary argument;
Prompt for number of students and store this in Limit;
For (x := 1 to Limit) do the following

```
            currentStud.inputData(x);
            studentTree.insert(studentTree.root, currentStud);
        End-For;
STOP
        E3.The void inputAlumni ( ) Method
START
        Let x, Limit be integers;
        Let currentAlumni be an instance of Generic;
        currentAlumni.setDataType(3);
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        Instantiate currentAlumni using dummyC as an intermediary argument;
        Prompt for number of Alumnus and store this in Limit;
        For (x := 1 to Limit) do the following
            currentAlumni.inputData(x);
            AlumnusTree.insert(AlumnusTree.root, currentAlumni);
        End-For;
STOP
        E4.The void inputEmployees ( ) Method
START
        Let x, Limit be integers;
        Let currentEmployee be an instance of Generic;
        currentEmployee.setDataType(1);
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        Instantiate currentEmployee using dummyC as an intermediary argument;
        Prompt for number of Employees and store this in Limit;
        For (x := 1 to Limit) do the following
            currentEmployee.inputData(x);
            employeeTree.insert(employeeTree.root, currentEmployee);
        End-For;
STOP
        E5. The void queryStudents() Method
START
        Let searchArg be an integer;
        Let foundStud be a GenericNode object;
        Let soughtStud be a Generic object;
        Set soughtStud.setDataType.(2)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT studentTree.isEmpty())
            While (User wishes to continue) do the following:
                foundStud := NULL;
                Prompt for the student's ID and store this in searchArg;
```

```
                    Instantiate soughtStud using dummyC as the intermediary parameter;
                    soughtStud.mID_Number := searchArg;
                    foundStud := studentTree.directSearch (soughtStud);
                    If (foundStud <> NULL)
                        Display (foundStud.nInfo.printMe( ));
                    Else Display ("Student is not in the tree");
                    End-If
                    Find out if the user wishes to continue;
                End-While;
        End-If
STOP
        E6.The void queryAlumni() Method
START
        Let searchArg be an integer;
        Let foundAlumnus be a GenericNode object;
        Let soughtAlumnus be a Generic object;
        Set soughtAlumnus.setDataType.(3)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT alumnusTree.isEmpty())
            While (User wishes to continue) do the following:
                foundAlumnus := NULL;
                Prompt for the Alumnus' ID and store this in searchArg;
                Instantiate soughtAlumnus using dummyC as the intermediary parameter;
                soughtAlumnus.mID_Number := searchArg;
                foundAlumnus := alumnusTree.directSearch (soughtAlumnus);
                If (foundAlumnus <> NULL)
                    Display (foundAlumnus.nInfo.printMe( ));
                Else Display ("Alumni is not in the tree");
                End-If
                Find out if the user wishes to continue;
            End-While;
        End-If
STOP
        E7.The void queryEmployees() Method
START
        Let searchArg be an integer;
        Let foundEmployee be a GenericNode object;
        Let soughtEmployee be a Generic object;
        Set soughtEmployee.setDataType.(1)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT employeeTree.isEmpty())
            While (User wishes to continue) do the following:
```

```
                    foundEmployee := NULL;
                    Prompt for the Employee's ID and store this in searchArg;
                    Instantiate soughtEmployee using dummyC as the intermediary
parameter;
                    soughtEmployee.mID_Number := searchArg;
                    foundEmployee := employeeTree.directSearch (soughtEmployee);
                    If (foundEmployee <> NULL)
                        Display (foundEmployee.nInfo.printMe( ));
                    Else Display ("Employee is not in the tree");
                    End-If
                    Find out if the user wishes to continue;
                End-While;
        End-If
STOP
        E8.The void removeStudents() Method
START
        Let searchArg be an integer;
        Let foundStud be a GenericNode object;
        Let soughtStud be a Generic object;
        Set soughtStud.setDataType.(2)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT studentTree.isEmpty())
            While (User wishes to continue) do the following:
                    foundStud := NULL;
                    Prompt for the student's ID and store this in searchArg;
                    Instantiate soughtStud using dummyC as the intermediary parameter;
                    soughtStud.mID_Number := searchArg;
                    foundStud := BS_Tree1.directSearch (soughtStud);
                    If (foundStud <> NULL)
                        Display (foundStud.nInfo.printMe( ));
                        Alert the user that the entire subtree starting at the identified node
will be deleted;
                        Prompt the user to confirm the deletion request;
                        If (Confirmation obtained)
studentTree.removeSubtree(foundStud); End-If;
                    Else Display ("Student is not in the tree");
                    End-If
                    Find out if the user wishes to continue;
                    If (BS_Tree1.isEmpty())
                        Inform the user that the tree is empty;
                        Exit the loop;
                    End-If;
            End-While;
```

```
        End-If
STOP
        E9.The void removeAlumni() Method
START
        Let searchArg be an integer;
        Let foundAlumni be a GenericNode object;
        Let soughtAlumni be a Generic object;
        Set soughtAlumni.setDataType.(3)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT alumnusTree.isEmpty())
                While (User wishes to continue) do the following:
                        foundAlumni := NULL;
                        Prompt for the Alumni's ID and store this in searchArg;
                        Instantiate soughtAlumni using dummyC as the intermediary parameter;
                        soughtAlumni.mID_Number := searchArg;
                        foundAlumni := alumnusTree.directSearch (soughtAlumni);
                        If (foundAlumni <> NULL)
                                Display (foundAlumni.nInfo.printMe( ));
                                Alert the user that the entire subtree starting at the identified node
will be deleted;
                                Prompt the user to confirm the deletion request;
                                If (Confirmation obtained)
alumnusTree.removeSubtree(foundAlumni); End-If;
                        Else Display ("Alumni is not in the tree");
                        End-If
                        Find out if the user wishes to continue;
                        If (alumnusTree.isEmpty())
                                Inform the user that the tree is empty;
                                Exit the loop;
                        End-If;
                End-While;
        End-If
STOP
        E10.The void removeEmployees() Method
START
        Let searchArg be an integer;
        Let foundEmployee be a GenericNode object;
        Let soughtEmployee be a Generic object;
        Set soughtEmployee.setDataType.(`)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT employeeTree.isEmpty())
                While (User wishes to continue) do the following:
```

```
                    foundEmployee := NULL;
                    Prompt for the Employee's ID and store this in searchArg;
                    Instantiate soughtEmployee using dummyC as the intermediary
parameter;
                    soughtEmployee.mID_Number := searchArg;
                    foundEmployee := employeeTree.directSearch (soughtEmployee);
                    If (foundEmployee <> NULL)
                        Display (foundEmployee.nInfo.printMe( ));
                        Alert the user that the entire subtree starting at the identified node
will be deleted;
                        Prompt the user to confirm the deletion request;
                        If (Confirmation obtained)
employeeTree.removeSubtree(foundEmployee); End-If;
                    Else Display ("Employee is not in the tree");
                    End-If
                    Find out if the user wishes to continue;
                    If (employeeTree.isEmpty())
                        Inform the user that the tree is empty;
                        Exit the loop;
                    End-If;
                End-While;
        End-If
STOP
        E11.The void modifyStudents() Method
START
        Let searchArg be an integer;
        Let foundStud be a GenericNode object;
        Let soughtStud be a Generic object;
        Set soughtStud.setDataType.(2)
        Let revisedStud be a Generic object;
        Set revisedStud.setDataType.(2)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT studentTree.isEmpty())
            While (User wishes to continue) do the following:
                    foundStud := NULL;
                    Prompt for the student's ID and store this in searchArg;
                    Instantiate soughtStud using dummyC as the intermediary parameter;
                    soughtStud.mID_Number := searchArg;
                    foundStud := studentTree.directSearch (soughtStud);
                    If (foundStud <> NULL)
                        Display (foundStud.nInfo.printMe( ));
                        Prompt the user to confirm the modification request;
                        If (Confirmation obtained)
```

```
                                        Instantiate revisedStud using dummyC as the intermediary
parameter;
                                        revisedStud.inputData(1);
                                        foundStud.nInfo.modifyMe(revisedStud);
                                End-If;
                        Else Display (“Student is not in the tree”);
                        End-If
                        Find out if the user wishes to continue;
                End-While;
        End-If
STOP
        E12.The void modifyAlumni() Method
START
        Let searchArg be an integer;
        Let foundAlumni be a GenericNode object;
        Let soughtAlumni be a Generic object;
        Set soughtAlumni.setDataType.(3)
        Let revisedAlumni be a Generic object;
        Set revisedAlumni.setDataType.(3)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT alumnusTree.isEmpty())
                While (User wishes to continue) do the following:
                        foundAlumni := NULL;
                        Prompt for the Alumni’s ID and store this in searchArg;
                        Instantiate soughtAlumni using dummyC as the intermediary parameter;
                        soughtAlumni.mID_Number := searchArg;
                        foundAlumni := alumnusTree.directSearch (soughtAlumni);
                        If (foundAlumni <> NULL)
                                Display (foundAlumni.nInfo.printMe( ));
                                Prompt the user to confirm the modification request;
                                If (Confirmation obtained)
                                        Instantiate revisedAlumni using dummyC as the
intermediary parameter;
                                        revisedAlumni.inputData(1);
                                        foundAlumni.nInfo.modifyMe(revisedAlumni);
                                End-If;
                        Else Display (“Alumni is not in the tree”);
                        End-If
                        Find out if the user wishes to continue;
                End-While;
        End-If
STOP
        E13.The void modifyEmployees() Method
```

```
START
        Let searchArg be an integer;
        Let foundEmployee be a GenericNode object;
        Let soughtEmployee be a Generic object;
        Set soughtEmployee.setDataType.(1)
        Let revisedEmployee be a Generic object;
        Set revisedEmployee.setDataType.(1)
        Let dummyC be an instance of CollegeMember;
        Instantiate dummyC;
        If (NOT employeeTree.isEmpty())
                While (User wishes to continue) do the following:
                        foundEmployee := NULL;
                        Prompt for the Employee's ID and store this in searchArg;
                        Instantiate soughtEmployee using dummyC as the intermediary
parameter;
                        soughtEmployee.mID_Number := searchArg;
                        foundEmployee := employeeTree.directSearch (soughtAlumni);
                        If (foundEmployee <> NULL)
                                Display (foundEmployee.nInfo.printMe( ));
                                Prompt the user to confirm the modification request;
                                If (Confirmation obtained)
                                        Instantiate revisedEmployee using dummyC as the
intermediary parameter;
                                        revisedEmployee.inputData(1);
                                        foundEmployee.nInfo.modifyMe(revisedEmployee);
                                End-If;
                        Else Display ("Employee is not in the tree");
                        End-If
                        Find out if the user wishes to continue;
                End-While;
        End-If
STOP
        E14.The void traverseStudents() Method
START
        Let x be an integer;
        Let outputS be a string;
        If (NOT studentTree.isEmpty())
                studentTree. inOrderPrep(); // Prepare for inOrder traversal
                studentTree.inOrderTraversal(BS_Tree1.root);
                For (x := 1 to studentTree.getSize() with increments of 1) do the following
                        Append studentTree.travResult.get(x - 1).printMe() + to outputS;
                End-For;
                Display ("The in-order traversal of the students tree is as follows: " + + outputS);
        End-If;
```

```
STOP
        E15.The void traverseAlumni() Method
START
        Let x be an integer;
        Let outputS be a string;
        If (NOT alumnusTree.isEmpty())
                alumnusTree. inOrderPrep(); // Prepare for inOrder traversal
                alumnusTree.inOrderTraversal(BS_Tree1.root);
                For (x := 1 to alumnusTree.getSize() with increments of 1) do the following
                        Append alumnusTree.travResult.get(x - 1).printMe() + to outputS;
                End-For;
                Display (“The in-order traversal of the alumnus tree is as follows: “ + + outputS);
        End-If;
STOP
        E16.The void traverseEmployees() Method
START
        Let x be an integer;
        Let outputS be a string;
        If (NOT employeeTree.isEmpty())
                employeeTree. inOrderPrep(); // Prepare for inOrder traversal
                employeeTree.inOrderTraversal(BS_Tree1.root);
                For (x := 1 to employeeTree.getSize() with increments of 1) do the following
                        Append employeeTree.travResult.get(x - 1).printMe() + to outputS;
                End-For;
                Display (“The in-order traversal of the employee tree is as follows: “ + + outputS);
        End-If;
STOP
        E17.The void checkTreeSize() Method
START
        Let k be an integer;
        Let quitTime be boolean, initialized to False;
        While (NOT quitTime)
                Present the user with the following menu:
                1. Check Students Tree
                2. Check Alumni Tree
                3. Check Employees Tree
                4. Quit;
                Prompt user to key in the a menu selection and store this in k;
                Case k is:
                        1: Display(“The size of the students tree is “ + BS_Tree1.getSize());
                        2: Display(“The size of the alumni tree is “ + BS_Tree2.getSize());
                        3: Display(“The size of the employees tree is “ + BS_Tree3.getSize());
                        4: quitTime := True;
                        Otherwise: Display(“Invalid choice”);
```

```
            End-Case;
        End-While;
STOP
        E18.The void emptyTree() Method
START
        Let k be an integer;
        Let quitTime be boolean, initialized to False;
        While (NOT quitTime)
            Present the user with the following menu;
            1. Empty Students Tree
            2. Empty Alumni Tree
            3. Empty Employees Tree
            4. Quit;
        Prompt user to key in the a menu selection and store this in k;
            Case k is:
            1: BS_Tree1.clearTree(); Display("The students tree has been cleared");
            2: BS_Tree2.clearTree(); Display("The alumni tree has been cleared");
            3: BS_Tree3.clearTree(); Display("The employees tree has been cleared");
            4: quitTime := True;
            Otherwise: Display("Invalid choice");
            End-Case;
        End-While;
STOP
        E19.The void initialize() Method
START
        Instantiate employeeTree, studentTree, alumnusTree;
STOP
```