Mustafa Tayyip BAYRAM
257639

Homework 3

Dependences between classes and objects, trees service

The various dependencies can exists between objects in the program. One of them is the so-called aggregation (whole - part relationship), which may be strong or weak. In the case of strong aggregation, the lifetime of object parts of the object is contained in the lifetime of the whole. Usually, when strong agregation occurs, parts are destroyed along with the aggregate. In the case of weak aggression, the lifetimes of parts and the whole object (agregate) are independent. The parts can "survive" the object lifetime, it can happens when the parts are shared by different aggregates, for example.

One of the concepts used in programming are the so-called tree. They are built of many objects called nodes. The tree is a special case of a graph. The concept of tree in programming can be used for solving different problems. For example, trees can be useful in sorting, information retrieval, algebraic expression processing, natural language analysis, etc.

The aim of the homework is to familiarized with using the concept of tree in programming.

Your tasks are below:

1. Implement class `CNodeStatic`, whose objects will represent the nodes in the tree.

Requirements for the `CNodeStatic` class, example interface:

```
class  CNodeStatic
{
public:
      CNodeStatic() {i_val = 0;};
      ~CNodeStatic();

      void  vSetValue(int  iNewVal) {i_val = iNewVal;};

      int  iGetChildrenNumber() {return(v_children.size());};
      void  vAddNewChild();
      CNodeStatic * pcGetChild(int  iChildOffset);

      void  vPrint() {cout << " " << i_val;};
      void  vPrintAllBelow();
private:
      vector<CNodeStatic>  v_children;
      int  i_val;
}//class  CNodeStatic
```

• The `vAddNewChild()` method adds a new child to the children vector of the node
• The `pcGetChild(int iChildOffset)` method, returns a child with given offset, if the offset is incorrect then returns NULL

• The `vPrintAllBelow()` method performs the `vPrint()` method for a given node, and `vPrintAllBelow()` for subordinate nodes (children), which leads to displaying all for node and all children values down the tree

2. Using class `CNodeStatic`, build a tree with several levels and test its operation. An example is below:

```
void  v_tree_test()
{
      CNodeStatic  c_root;

      c_root.vAddNewChild();
      c_root.vAddNewChild();

      c_root.pcGetChild(0)->vSetValue(1);
      c_root.pcGetChild(1)->vSetValue(2);

      c_root.pcGetChild(0)->vAddNewChild();
      c_root.pcGetChild(0)->vAddNewChild();

      c_root.pcGetChild(0)->pcGetChild(0)->vSetValue(11);
      c_root.pcGetChild(0)->pcGetChild(1)->vSetValue(12);


      c_root.pcGetChild(1)->vAddNewChild();
      c_root.pcGetChild(1)->vAddNewChild();

      c_root.pcGetChild(1)->pcGetChild(0)->vSetValue(21);
      c_root.pcGetChild(1)->pcGetChild(1)->vSetValue(22);
}//void  v_tree_test()
```

Consider what the destructor should look like so that all nodes are removed.

3. For the `CNodeStatic` class, implemnt the `vPrintUp()` method, which for any node will display the values of all its parents, from the selected node to the root. For example, for the tree from the example above, the following "text":

   ```
   c_root.pcGetChild(0)->pcGetChild(1)->vPrintUp();
   ```
   will display 12 1 0

Think about what data is needed in the CNodeStatic class to perform the method described above.

We should store the parent pointer on CNodeStatic class for point parent of nodes

4. The node should not be a class with which the program communicates directly. Wrap it with a tree in `CTreeStatic` class.

```
class  CTreeStatic
{
public:
      CTreeStatic();
      ~CTreeStatic();

      CNodeStatic  * pcGetRoot()  {return(&c_root);}
      void  vPrintTree();
private:
      CNodeStatic  c_root;
```

```
}//class  CTreeStatic
```

The `vPrintTree()` method displays the values of all nodes to the screen.

Consider whether the `CTreeStatic` and `CNodeStatic` classes should be in the same or separate source files (cpp & h).

Actually it doesn't matter, both will work, but separate files makes it more organized.

5. Implement the `CnodeDynamic` and `CTreeDynamic` classes, which instead of storing child static nodes will store dynamic pointers.

```
class  CNodeDynamic
{
public:
      CNodeStatic() {i_val = 0;};
      ~CNodeStatic();
      void  vSetValue(int  iNewVal) {i_val = iNewVal;};

      int  iGetChildrenNumber() {return(v_children.size);};
      void  vAddNewChild();
      CTreeDynamic  *pcGetChild(int  iChildOffset);

      void  vPrint() {cout << " " << i_val;};
      void  vPrintAllBelow();
private:
      vector<CNodeDynamic *>  v_children;
      int  i_val;
}//class  CNodeDynamic

class  CTreeDynamic
{
public:
      CTreeDynamic();
      ~CTreeDynamic();

      CTreeDynamic  *pcGetRoot()  {return(&c_root);}
      void  vPrintTree();
private:
      CTreeDynamic  *pc_root;
}//class  CTreeStatic
```

Should the `CTreeDynamic`, `CNodeDynamic` and `CTreeStatic`, `CNodeStatic` classes destructors differ? Yes it must be different because we are dealing with pointers in Dynamic classes. There is no need to creating destructors because we are not using dynamic memory allocation.

6. Implement the following method for the tree class.

```
bool    bMoveSubtree(CNode  *pcParentNode,  CNode  *pcNewChildNode,  CNode  *
pc2ParentNode);
```

The `bMoveSubtree` method accepts three nodes as parameters. The first is the node from the tree for which it was called, and the second one from the "alien" tree. The purpose of the method is to remove the subtree of the `pcNewChildNode` node from the tree it was in before ("alien" tree) and insert it as the "child" of the tree where it is now to be found. The `pc2ParentNode`

parameter is needed to remove a subtree from a "alien" tree. The trees before and after the operation are shown in Figure 1.

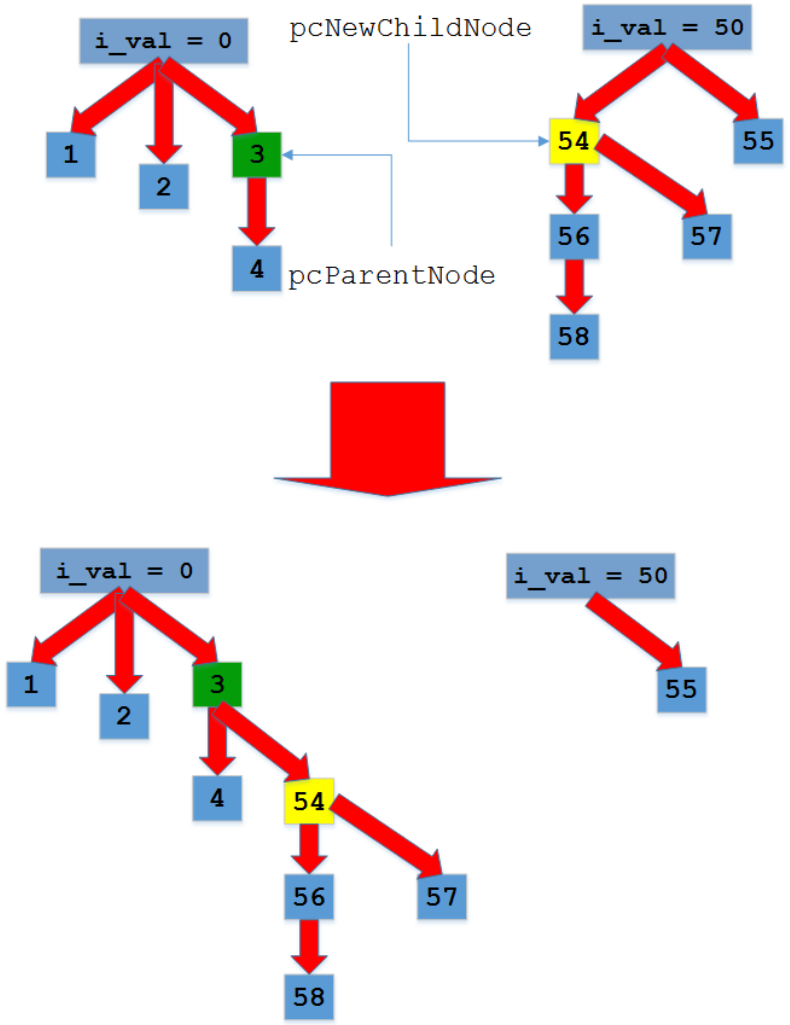

Figure 1. An example of moving a subtree between trees

Some questions:

1. **Will the bMoveSubtree method be easily used for the CTreeDynamic, CNodeDynamic classes and for the CTreeStatic, CNodeStatic classes?**
   Yes it can be use easily.
2. **Consider whether storing lists of statically allocated objects has more advantages or disadvantages.**
   Storing dynamic has more advantages about effective programming. Because pointers memory is less than objects memory.