Software Engineering
Lab 7 Report

Emre Ozan Alkan
{emreozanalkan@gmail.com}
MSCV-5

22 November 2013

1 Binary Tree Dictionary

In this lab we studied binary trees, and how to recursively construct, traverse, print and delete it. We created example dictionary application as requested in lab paper.

1.1 Node Class

Node class declared and implemented to store char array as pointer, representing a word in dictionary tree.

Listing 1: Node.h

```cpp

class Node
{
private:
    char* _data;          // The data in this node
    Node* _leftNode;      // Pointer to the left node
    Node* _rightNode;     // Pointer to the right node
public:
    Node();
    Node(char*);
    ~Node();

    void setData(char*);          // _data setter
    void setLeftNode(Node*);      // _left setter
    void setRightNode(Node*);     // _right setter

    char* getData() const;            // _data getter
    Node* getLeftNode() const;        // _left getter
    Node* getRightNode() const;       // _right getter

    void insertData(char* data);      // Inserts data to the tree

    void printPreOrder() const;     // Prints the tree Pre-Order
    void printPostOrder() const;    // Prints the tree Post-Order
    void printInOrder() const;      // Prints the tree In-Order
};
```

1.2 Binary Tree Construction

I created insertion function to construct and insert data to tree in lexical order as requested.

Listing 2: Tree Insertion

```

// Getting word as char array, and inserting it to correct poisition in tree.
// If it's data is empty, stores in it self, otherwise compare with its data
// to check lexical order, leaxically smaller word tend to go left, others
// inserted to right in binary data structure context.
void Node::insertData(char* data)
{
    if(this->_data == 0)
    {
        this->_data = data;
        return;
    }

    int compareResult = strcmp(this->_data, data);

    if(compareResult == 0)
        return;
    else if(compareResult > 0)
    {
        if(this->_leftNode == 0)
        {
            Node* leftNode = new Node(data);
            this->setLeftNode(leftNode);
        }
        else
            this->_leftNode->insertData(data);

    }
    else
    {
        if(this->_rightNode == 0)
        {
            Node* rightNode = new Node(data);
            this->setRightNode(rightNode);
        }
        else
            this->_rightNode->insertData(data);
    }
}
```

1.3 Printing Binary Tree

Here is the 3 displaying/printing function for our binary tree, which are traversing Pre-Order, Post-Order and In Order.

Listing 3: Printing Tree

```

// The Pre-Order traversal: at each node the root is evaluated first
// then the left sub tree, the the right subtree.
void Node::printPreOrder() const
{
    if(this->_data != 0)
        cout<<"word="<<this->_data<<endl;

    if(this->_leftNode != 0)
        this->_leftNode->printPreOrder();

    if(this->_rightNode != 0)
```

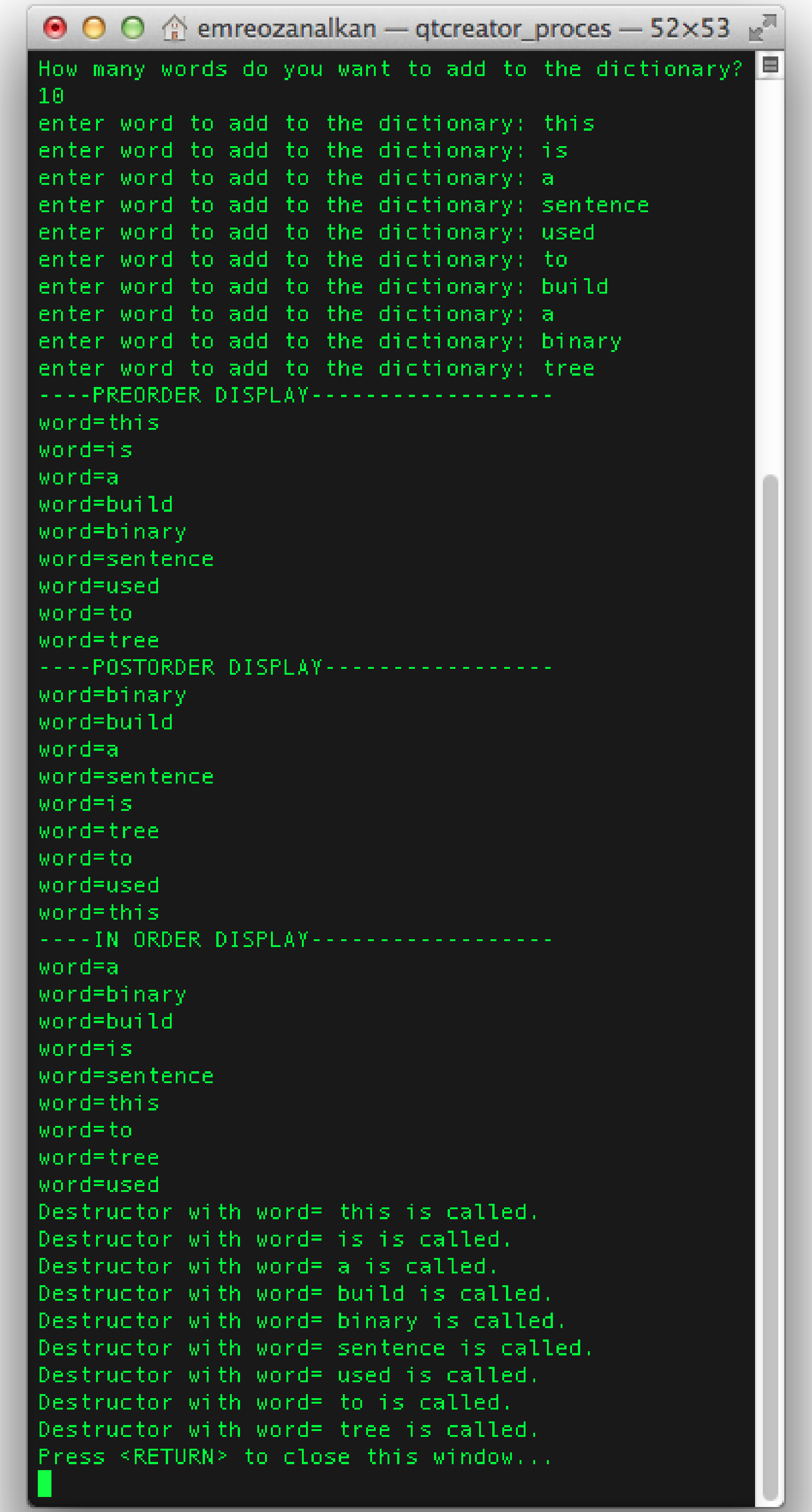
```
        this->_rightNode->printPreOrder();
}

// The Post-Order traversal: the left subtree first
// then the right subtree, then the root
void Node::printPostOrder() const
{
    if(this->_leftNode != 0)
        this->_leftNode->printPostOrder();

    if(this->_rightNode != 0)
        this->_rightNode->printPostOrder();

    if(this->_data != 0)
        cout<<"word="<<this->_data<<endl;
}

// The In Order traversal: left, root, then right nodes evaluated.
void Node::printInOrder() const
{
    if(this->_leftNode != 0)
        this->_leftNode->printInOrder();

    if(this->_data != 0)
        cout<<"word="<<this->_data<<endl;

    if(this->_rightNode != 0)
        this->_rightNode->printInOrder();
}
```

1.4 Example Main

Listing 4: Node Example Main

```

int main(void)
{
    Node* rootNode = new Node();
    char* word = 0;
    int wordCount = 0;

    cout<<"How many words do you want to add to the dictionary?"<<endl;
    cin>>wordCount;

    for(int i = 0; i < wordCount; i++)
    {
        word = new char[256];
        cout<<"enter word to add to the dictionary: ";
        cin>>word;
        rootNode->insertData(word);
    }

    cout<<"———PREORDER DISPLAY—————————————"<<endl;
    rootNode->printPreOrder();
    cout<<"———POSTORDER DISPLAY————————————"<<endl;
    rootNode->printPostOrder();
    cout<<"———IN ORDER DISPLAY—————————————"<<endl;
    rootNode->printInOrder();

    delete rootNode;
    rootNode = 0;
    word = 0;

    return 0;
}
```

1.5 Result



```
How many words do you want to add to the dictionary?
10
enter word to add to the dictionary: this
enter word to add to the dictionary: is
enter word to add to the dictionary: a
enter word to add to the dictionary: sentence
enter word to add to the dictionary: used
enter word to add to the dictionary: to
enter word to add to the dictionary: build
enter word to add to the dictionary: a
enter word to add to the dictionary: binary
enter word to add to the dictionary: tree
----PREORDER DISPLAY-----------------
word=this
word=is
word=a
word=build
word=binary
word=sentence
word=used
word=to
word=tree
----POSTORDER DISPLAY----------------
word=binary
word=build
word=a
word=sentence
word=is
word=tree
word=to
word=used
word=this
----IN ORDER DISPLAY-----------------
word=a
word=binary
word=build
word=is
word=sentence
word=this
word=to
word=tree
word=used
Destructor with word= this is called.
Destructor with word= is is called.
Destructor with word= a is called.
Destructor with word= build is called.
Destructor with word= binary is called.
Destructor with word= sentence is called.
Destructor with word= used is called.
Destructor with word= to is called.
Destructor with word= tree is called.
Press <RETURN> to close this window...
```