```
                #####   #####   #####
#####    #   # #     # #     # #     #
#    #    # #  #     #       #       #
#####      #    ####### #####   #####
#    #     #         #       # #
#    #     #   #     # #     # #
#####      #    #####   #####  #######
```

Nov 9 11:52 2014 by932/binarytree.cpp Page 1

```
/********************************
*Filename:binarytree.cpp        *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include "binarytree.h"
using namespace std;

int TreeDataCmp(const linked_list& first, const linked_list& second)
{
        if (first->line_num < second->line_num)
                return -1;
        if (first->line_num == second->line_num)
                return 0;
        if (first->line_num > second->line_num)
                return 1;
        return 0;
}

BinaryTree::BinaryTree()
{
        root = NULL;
}
BinaryTree::~BinaryTree()
{
        postorderdelete(root);
}
void BinaryTree::postorderdelete(TreeNode* curr)
{
        if (curr != NULL)
        {
                postorderdelete(curr->left);
                postorderdelete(curr->right);

                delete curr;
        }
}
void BinaryTree::SetIterator()
{
        curr = root;
        if (curr != NULL)
                while (curr->left != NULL)
                        curr = curr->left;
}

linked_list BinaryTree::Next()
{
        linked_list rvalue = curr->data;

        if (curr->right != NULL)        // into the right subtree
        {
                curr = curr->right;
                while (curr->left != NULL)
                        curr = curr->left;
```

```
		}
		else if (curr->parent == NULL)  // no parent
			curr = NULL;            // we are finished
		else                            // find ancestor
		{
			while (curr->parent != NULL
					&& (curr->parent)->left != curr)
				curr = curr->parent;
			curr = curr->parent;
		}
		return rvalue;
	}


	bool BinaryTree::More()
	{
		if (curr != NULL)
			return true;
		else
			return false;
	}



	void BinaryTree::Insert(const linked_list& data)
	{
		InsertNode(data, root);
	}

	bool BinaryTree::Locate(const linked_list& data, linked_list& founddata)
	{
		TreeNode* foundNode = FindNode(data, root);

		if (foundNode != NULL)
		{
			founddata = foundNode->data;
			return true;
		}
		else
			return false;

	}

	bool BinaryTree::Delete(const linked_list& data, linked_list& deldata)
	{
		TreeNode *parent, *prev, *curr = FindNode(data, root);

		if (curr == NULL)
			return false;
		deldata = curr->data;
		parent = curr->parent;
		if (curr->left == NULL)                 // no left subtree
		{
			if (curr->right == NULL)        // no children
			{
				if (parent == NULL)     // curr is root
```

```
                                root = NULL;
                        else if (parent->left == curr)
                                parent->left = NULL;
                        else
                                parent->right = NULL;
                }
                else                            // only right subtree
                {
                        if (parent == NULL)
                        {
                                root = curr->right;
                                root->parent = NULL;
                        }
                        else if (parent->left == curr)
                        {
                                parent->left = curr->right;
                                curr->right->parent = parent;
                        }
                        else
                        {
                                parent->right = curr->right;
                                curr->right->parent = parent;
                        }
                }
        }
        else if (curr->right == NULL)           // only left subtree
        {
                if (parent == NULL)
                {
                        root = curr->left;
                        root->parent = NULL;
                }
                else if (parent->left == curr)
                {
                        parent->left = curr->left;
                        curr->left->parent = parent;
                }
                else
                {
                        parent->right = curr->left;
                        curr->left->parent = parent;
                }
        }
        else                            // has both subtrees
        {
                prev = curr->left;
                while (prev->right != NULL)
                        prev = prev->right;
                prev->right = curr->right;
                curr->right->parent = prev;
                if (parent == NULL)
                {
                        root = curr->left;
                        root->parent = NULL;
                }
                else
```

```
			{
				parent->right = curr->left;
				curr->left->parent = parent;
			}
		}
		delete curr;
		return true;
	}


	BinaryTree::TreeNode* BinaryTree::FindNode(const linked_list& data, BinaryTree::TreeNode* tree_root)
	{
		if (tree_root == NULL)
		{
			return 0;
		}
		int Result = TreeDataCmp(data,tree_root->data);

		if (Result == 0)
			return tree_root;
		if (Result < 0 && tree_root->left != NULL)
			return FindNode(data,tree_root->left);
		else if (Result > 0 && tree_root->right != NULL)
			return FindNode(data,tree_root->right);
		return 0;
	}


	void BinaryTree::InsertNode(const linked_list& data, BinaryTree::TreeNode*& tree_root)
	{
		//insert root node
		if (tree_root == NULL)
		{
			tree_root = new TreeNode;	//set up a new node
			tree_root->data = data;
			tree_root->left = NULL;
			tree_root->right = NULL;
			tree_root->parent = NULL;
		}
		else if (TreeDataCmp(data,tree_root->data) <= 0)
		{
			if (tree_root->left == NULL)
			{
				tree_root->left = new TreeNode;
				tree_root->left->data = data;
				tree_root->left->left = NULL;
				tree_root->left->right = NULL;
				tree_root->left->parent = tree_root;
			}
			else
				InsertNode(data,tree_root->left);
		}
		else
		{
			if (tree_root->right == NULL)
			{
```

```
                                tree_root->right = new TreeNode;
                                tree_root->right->data = data;
                                tree_root->right->left = NULL;
                                tree_root->right->right = NULL;
                                tree_root->right->parent = tree_root;
                        }
                        else
                                InsertNode(data,tree_root->right);
                }
        }

```

```
Nov  9 11:52 2014  by932/linkedlist.cpp Page 1


/********************************
*Filename:linkedlist.cpp        *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include "linkedlist.h"
using namespace std;
int listdatacmp(const T& a, const T& b)                 // user defined compare function
{
                // return < 0 if a < b
                // return == 0 if a == b
                // return >0 if a > b;

                return 0;
}

// iterator methods

void linkedlist::setiterator()
{
    iterator_current = head;
}

// is there more in the iterator

bool linkedlist::more()
{
    if (iterator_current != NULL)
        return true;
    else
        return false;
}

// get next piece of data out of iterator

T linkedlist::next()
{
    T tmp = iterator_current->data;
    iterator_current = iterator_current->next;
    return tmp;
}

void linkedlist::insertbeforecurrent(const T& newdata)                  // insertion method
{
        nodeptr tmp;
        tmp = new node;
        tmp->data = newdata;
        tmp->next = NULL;

        nodeptr curr;
        nodeptr prev;
        curr = head;
        prev = NULL;

```

```
		while (curr != iterator_current)
		{
			prev = curr;
			curr = curr->next;
		}

		if (prev == NULL)
			head = tmp;
		else
			prev->next = tmp;
		tmp->next = curr;
	}

	bool linkedlist::deletecurrent(T& retdata)
	{
		nodeptr curr;
		nodeptr prev;
		curr = head;
		prev = NULL;

		if (iterator_current == NULL)
		    return false;

		while (curr != iterator_current)                // find it
		{
			prev = curr;
			curr = curr->next;
		}

		if (curr == NULL)
		{
			return false;
		}

		if (prev)
		{
			prev->next = curr->next;
			retdata = curr->data;
			delete curr;
			return true;
		}
		else
		{
			head = curr->next;
			retdata = curr->data;
			delete curr;
			return true;
		}
	}

	// constructor
	linkedlist::linkedlist()
	{
		head = NULL;
	}
	//deconstructor
```

```
Nov  9 11:52 2014  by932/linkedlist.cpp Page 3


        linkedlist::~linkedlist()
        {
                node* temp = head;
                while(temp && temp->next)
                {
                        temp=temp->next;
                        delete temp;    //delet linked-list
                }
        }
        // add to tail method
        void linkedlist::addtotail(char token[], int line_no)
        {
                nodeptr tmp, curr;
                tmp = new node;
                tmp->data.content = token;
                tmp->data.line_num = line_no;
                tmp->next = NULL;

                if (head != NULL)
                {
                        curr = head;
                        while (curr->next)
                                curr = curr->next;
                        curr->next = tmp;
                }
                else
                        head = tmp;
        }

        // check if list is empty method
        bool linkedlist::isempty()
        {
                if (head == NULL)
                        return true;
                else
                        return false;
        }

        // remove from head method
        T linkedlist::removefromhead()
        {
                nodeptr tmp;
                T data;

                data = head->data;
                tmp = head;
                head = head->next;
                delete tmp;
                return data;
        }

        void linkedlist::insert(const T& newdata)                       // insertion method
        {
                nodeptr tmp;
                tmp = new node;
                tmp->data = newdata;
```

```
Nov  9 11:52 2014  by932/linkedlist.cpp Page 4


                tmp->next = NULL;

                nodeptr curr;
                nodeptr prev;
                curr = head;
                prev = NULL;

                while (curr && listdatacmp(newdata, curr->data) >= 0)
                {
                        prev = curr;
                        curr = curr->next;
                }

                if (prev == NULL)
                        head = tmp;
                else
                        prev->next = tmp;
                tmp->next = curr;


        }

        bool linkedlist::locate(const T& keydata, T& retresult)         // locate method
        {
                nodeptr curr = head;

                while (curr && listdatacmp(keydata, curr->data) != 0)
                {
                        curr = curr->next;
                }

                if (curr == NULL)
                        return false;
                else
                {
                        retresult = curr->data;
                        return true;
                }
        }

        bool linkedlist::delete_node(const T& keydata, T& retdata)              // delete node method
        {
                nodeptr curr;
                nodeptr prev;
                curr = head;
                prev = NULL;

                while (curr != NULL && listdatacmp(keydata, curr->data) != 0)
                {
                        prev = curr;
                        curr = curr->next;
                }

                if (curr == NULL)
                {
                        return false;
```

```
		}

		if (prev)
		{
			prev->next = curr->next;
			retdata = curr->data;
			delete curr;
			return true;
		}
		else
		{
			head = curr->next;
			retdata = curr->data;
			delete curr;
			return true;
		}
	}
	bool linkedlist::print(ostream&)
	{
		if (head == NULL)
		{
			return false;
		}
		nodeptr temp = head;				// do not change the head pointer
		while (temp != NULL)
		{
			cout << "token = " << temp->data.content << '\t' << "line_num = " << temp->data.line_num;
			cout << endl;
			temp = temp->next;
		}
		return true;

	}
```

```
Nov  9 11:52 2014  by932/main.cpp Page 1


/********************************
*Filename:main.cpp              *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include "program.h"
using namespace std;

int main()
{
        program test_program_class;
        LIST test_list_class;

        test_program_class.getline(test_list_class);
        test_list_class.print(cout);
        return 0;
}
```

```
Nov  9 11:52 2014  by932/program-list.cpp Page 1
```

```
/********************************
*Filename:program-list.cpp      *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include "program-list.h"
using namespace std;
// constructor
LIST::LIST()
{
        head = NULL;

}

//deconstructor
LIST::~LIST()
{
        node* temp = head;
        while(temp && temp->next)
        {
                temp=temp->next;
                delete temp;    //delet linked-list
        }
}

bool LIST::load(char token[], int line)
{

        if (head == NULL)       // linked list is empty - so this will be the head node
        {

                NPtr newnode = new node;

                if (newnode == NULL)            // could not allocate memory
                {
                        cout << "Allocation error occured" << endl;
                        return false;
                }
                strcpy(newnode->content, token);
                newnode->line = line;
                newnode->next = NULL;
                head = newnode;

        }
        else            // if not here then the linked list exists
        {
                NPtr check = head;
                while (check != NULL)
                {
                        if (check->next == NULL)
                        {
                                NPtr newnode = new node;
                                if (newnode == NULL)
                                {
```

Nov 9 11:52 2014 by932/program-list.cpp Page 2

```
                                                cout << "Allocation error occured" << endl;
                                                return false;
                                        }
                                        strcpy(newnode->content, token);        //store token
                                        newnode->line = line;   //give the line number
                                        newnode->next = NULL;   //new node next node = null
                                        check->next = newnode;
                                        break;
                                }
                                check = check->next;

                        }
                }
                return true;

        }

        bool LIST::print(ostream&)
        {
                cout << "Start output Part 1" << endl;
                if (head == NULL)
                {
                        return false;
                }
                NPtr temp = head;                               // do not change the head pointer
                while (temp != NULL)
                {
                        //cout << "token = " << temp->content << '\t' << "line_num = " << temp->line;
                        //cout << endl;
                        temp = temp->next;
                }
                cout << "End output Part 1" << endl;
                return true;

        }
```

```
Nov  9 11:52 2014  by932/program.cpp Page 1


/********************************
*Filename:program.cpp           *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include <fstream>
#include <cstring>
#include "program.h"
#include "binarytree.h"
using namespace std;
//Global Data
BinaryTree WordTree;

bool bonus(char []);//delete 62 reserved words
program::program()
{
        line_no = 1;    //init line_no

}
program::~program()
{
        fin.close();    //close the file
}

bool program::open(char source_file[])
{

        fin.open(source_file);  //open file

        if(!fin)
        {
                cout << "Cann’t find file"<< endl;
                return false;   // open file false
        }
        return true;    //open file good

}

bool program::getline(LIST& obj)
{
        char fname[100], token[40], ch;
        int line_pos, carryon, i;

        cout << "Enter a filename: ";
        cin >> fname;

        bool good_or_bad = true;        //open file status good or bad, true is good
        good_or_bad = open(fname);

        if (good_or_bad == true)
        {
                line_pos = 0;
                token[0] = 0;           // initialise token

```

Nov 9 11:52 2014 by932/program.cpp Page 2

```
                        ch = fin.get();
                        while (!fin.eof())
                        {
                                //Eating all comments
                                if (ch == '/')
                                {
                                        ch = fin.get();
                                        line_pos++;
                                        //"//"type comment - eat both characters and all to newline
                                        if (ch == '/')
                                        {
                                                do
                                                {
                                                        ch = fin.get();
                                                } while (ch != '\n');
                                                line_no++;
                                                line_pos = 0;
                                                ch = fin.get(); // now positioned at start of next line
                                        }
                                        //old type comment/* */ - eat both characters and all to end
                                        else if (ch == '*')
                                        {
                                                carryon = 1;
                                                ch = fin.get();
                                                while (carryon)
                                                {
                                                        if (ch == '\n')
                                                        {
                                                                line_no++;
                                                                line_pos = 0;
                                                                ch = fin.get();
                                                        }
                                                        else if (ch == '*')
                                                        {
                                                                ch = fin.get();
                                                                if (ch == '/')
                                                                        carryon = 0;
                                                        }
                                                        else
                                                                ch = fin.get();
                                                }
                                                ch = fin.get();
                                                line_pos = 1;
                                        }
                                }
                                else if (ch == '"')
                                {
                                //character string constant - eat all characters to other end
                                //avoiding \anything
                                        ch = fin.get();
                                        while (ch != '"')
                                        {
                                                if (ch == '\\')
                                                        ch = fin.get();
                                                ch = fin.get();
                                        }
                                        ch = fin.get();
                                }
                                else if (ch == ’\’’)
                                {
                                        //character constant − as for character string
                                        ch = fin.get();
                                        while (ch != ’\’’)
                                        {
                                                if (ch == ’\\’)
                                                        ch = fin.get();
                                                ch = fin.get();
                                        }
                                        ch = fin.get();
                                }
                                else if (line_pos == 0 && ch == ’#’)
                                {
                                        //# pre−processor line − eat # and all to newline
                                        do
                                        {
                                                ch = fin.get();
                                        } while (ch != ’\n’);
                                        ch = fin.get();
                                        line_no++;
                                        line_pos = 0;
                                }
                                else if (isalpha(ch))   // start of token
                                {
                                        i = 0;
                                        do
                                        {

                                                token[i++] = ch;
                                                ch = fin.get();
                                        } while (isalnum(ch) || ch == ’_’);
                                        token[i] = 0;   //token end
                                        bool jud = true;
                                        jud = bonus(token);     //delete 62 reserved words
                                        if (jud == false)
                                                continue;
                                        //This is part 1
                                        obj.load(token,line_no);//store token and line number to the list

                                        //This is part 2
                                        bnode *temp = new bnode;
                                        temp−>content = token;
                                        temp−>line_num = line_no;
                                        temp−>data.addtotail(token, line_no);
                                        linked_list foundData;

                                        if (WordTree.Locate(temp, foundData))
                                        {
                                                //This part is store same line token to the linked−list
                                                foundData−>line_num = line_no;
                                                foundData−>data.addtotail(token, line_no);
                                                delete temp;
                                        }
                                        else
                                        {
                                                //This is set up a new tree node
                                                WordTree.Insert(temp);
                                        }


                                        token[0] = 0;   // initialise token
                                }
                                else if (ch == '\n')    // just a newline
                                {
                                        line_no++;
                                        line_pos = 0;
                                        ch = fin.get();
                                }
                                else
                                        ch = fin.get();
                        }
                }
                return true;

        }
        //this is part 3
        bool bonus(char token[])
        {
                if (strcmp(token, "asm") == 0 ||
                strcmp(token, "casecatch") == 0 ||
                strcmp(token, "const_cast") == 0 ||
                strcmp(token, "do") == 0 ||
                strcmp(token, "enum") == 0 ||
                strcmp(token, "float") == 0 ||
                strcmp(token, "if") == 0 ||
                strcmp(token, "mutable") == 0 ||
                strcmp(token, "private") == 0 ||
                strcmp(token, "reinterpret_cast") == 0 ||
                strcmp(token, "sizeof") == 0 ||
                strcmp(token, "switch") == 0 ||
                strcmp(token, "true") == 0 ||
                strcmp(token, "typename") == 0 ||
                strcmp(token, "virtual") == 0 ||
                strcmp(token, "while") == 0 ||
                strcmp(token, "auto") == 0 ||
                strcmp(token, "char") == 0 ||
                strcmp(token, "continue") == 0 ||
                strcmp(token, "double") == 0 ||
                strcmp(token, "explicit") == 0 ||
                strcmp(token, "for") == 0 ||
                strcmp(token, "inline") == 0 ||
                strcmp(token, "namespace") == 0 ||
                strcmp(token, "protected") == 0 ||
                strcmp(token, "return") == 0 ||
                strcmp(token, "static") == 0 ||
                strcmp(token, "template") == 0 ||
                strcmp(token, "try") == 0 ||
                strcmp(token, "union") == 0 ||
                strcmp(token, "void") == 0 ||
```

Nov 9 11:52 2014 by932/program.cpp Page 5

```
                strcmp(token, "bool") == 0 ||
                strcmp(token, "class") == 0 ||
                strcmp(token, "default") == 0 ||
                strcmp(token, "dynamic_cast") == 0 ||
                strcmp(token, "extern") == 0 ||
                strcmp(token, "friend") == 0 ||
                strcmp(token, "int") == 0 ||
                strcmp(token, "new") == 0 ||
                strcmp(token, "public") == 0 ||
                strcmp(token, "short") == 0 ||
                strcmp(token, "static_cast") == 0 ||
                strcmp(token, "this") == 0 ||
                strcmp(token, "typedef") == 0 ||
                strcmp(token, "unsigned") == 0 ||
                strcmp(token, "volatile") == 0 ||
                strcmp(token, "break") == 0 ||
                strcmp(token, "const") == 0 ||
                strcmp(token, "delete") == 0 ||
                strcmp(token, "else") == 0 ||
                strcmp(token, "false") == 0 ||
                strcmp(token, "goto") == 0 ||
                strcmp(token, "long") == 0 ||
                strcmp(token, "operator") == 0 ||
                strcmp(token, "register") == 0 ||
                strcmp(token, "signed") == 0 ||
                strcmp(token, "struct") == 0 ||
                strcmp(token, "throw") == 0 ||
                strcmp(token, "typeid") == 0 ||
                strcmp(token, "using") == 0 ||
                strcmp(token, "wchar_t") == 0)
                        return false;

                return true;

        }
```

Nov 9 11:52 2014 by932/binarytree.h Page 1

```
/********************************
*Filename:binarytree.h          *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include <cstring>
#include "linkedlist.h"
using namespace std;
// Definition of data contained in BinaryTree
struct bnode
{
        linkedlist data;        //store token
        int line_num;   //story line number
        char *content;  //token content[40];
};

typedef bnode * linked_listPtr;

typedef linked_listPtr linked_list;


class BinaryTree
{
        public:
                BinaryTree();//constructor
                ~BinaryTree();// destructor
                void Insert(const linked_list&);
                bool Locate(const linked_list&, linked_list&);
                bool Delete(const linked_list&, linked_list&);

                // iterator methods
                void SetIterator();             // find left most node
                linked_list Next();             // return next data item
                bool More();                    // are there any more nodes?

        private:
                struct TreeNode
                {
                        linked_list data;
                        TreeNode *left, *right, *parent;
                };

                TreeNode* root;
                TreeNode* curr;                         // used by iterator

                TreeNode* FindNode(const linked_list&, TreeNode*);
                void InsertNode(const linked_list&, TreeNode*&);
        //this function is given the root node and will do a post order deletion of the nodes
                void postorderdelete(TreeNode*);
};


```

```
Nov  9 11:52 2014  by932/linkedlist.h Page 1
```

```
/********************************
*Filename:linkedlist.h          *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
using namespace std;

struct linkedlist_node
{
        char *content;
        int line_num;   //story line number
};

typedef linkedlist_node T;



class linkedlist
{
        public: // public methods
                linkedlist();
                ~linkedlist();
                // the destructor has been removed − when you need to delete you must do by hand
                void addtotail(char [], int);
                bool print(ostream&);   //Print out the linked−list


                bool isempty();
                T removefromhead();
                void insert(const T&);
                bool locate(const T&, T&);
                bool delete_node(const T&, T&);

                bool deletecurrent(T&);
                void insertbeforecurrent(const T&);
                void setiterator();
                T next();
                bool more();
        private:        // node data and declaration − hidden in class
                struct node;
                typedef node* nodeptr;
                nodeptr iterator_current;
                struct node
                {
                        T data;
                        nodeptr next;
                };
                nodeptr head;
};

```

```
Nov  9 11:52 2014  by932/program-list.h Page 1


/********************************
*Filename:program-list.h        *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include <cstring>
using namespace std;
struct node
{
        char content[40];
        int line;
        node *next;
};
typedef node* NPtr;
class LIST
{
        public:
                LIST();
                ~LIST();
                bool load(char[], int);
                bool print(ostream&);
        private:
                NPtr head;    // pointer to list of chars
                NPtr next;
};
```

Nov 9 11:52 2014 by932/program.h Page 1

```
/********************************
*Filename:program.h             *
*Login:by932                    *
*AssignmentNo:ass5              *
*DateLastModified:2/11/2014     *
********************************/
#include <iostream>
#include <fstream>
#include <cstring>
#include "program-list.h"
using namespace std;

class program
{
	public:

		program();
		~program();
		bool open(char[]);			// open a file return state i.e. success
		bool getline(LIST&);

	private:
		ifstream fin;
		int line_no;
};


```

```
Enter a filename: Start output Part 1
End output Part 1
```