**Andrew Tran**

**Lab Experience Fourteen.**

**BST’s, Arrays, and Queues, and Miscellaneous Classes Lab**

**Due: Friday May 5 at the beginning of class.**

**What you must do:**

1. You are also to complete project 10.1 Version 2 in your lab manual as described below.

A text concordance is an alphabetical listing of all distinct words in a piece of text. You are to write a program to construct a concordance for a document stored in a file. The words are to be stored in an array of 26 binary search trees.

Your program should do the following:

1. Declares an array of 26 BST’s, one for each letter of the alphabet. Words that begin with an ‘a’ or ‘A’ will be stored in the BST at location zero of the array. Words that begin with ‘b’ or ‘B’ will be stored in the BST at location one. The pattern continues for the remaining letters of the alphabet.

**In general, words that begin with character ch will be stored in the array at location *upch* – ‘A’, where *upch* is the uppercase equivalent of ch.**

1. Read strings from a file and for each string:
   1. Removes all characters from it that are not letters and converts all letters to uppercase.
   2. If the string isn’t empty:
      1. Searches the appropriate BST for the string
         1. If the word already is in the BST, insert the line number where the word occurred into the Queue that is part of the node of the BST.
         2. If the word is not in the BST, insert the word and insert the line number where the word occurred into the Queue that is part of the node of the BST.
2. When the concordance is output, the words should be in alphabetical order, and for each word the line numbers will be displayed in ascending order.

**Hint:** You might want to create a class called either Word or Token where the attributes of the class is a string and an int. Your BST’s Nodes would be the class Queue and the class Word or Token. **Make sure all operators are appropriately overloaded for every class you create.**

For example.

Given the text file:

Ants and bats

and cows and cats

are animals

Cows are big

but ants are small

and cats are in

between

Concordance produced using the array name: concord

concord[0] ---------------------------> ANTS 🡪 1🡪5

AND ARE 🡪3🡪4🡪5🡪6

1🡪2🡪6

ANIMALS 🡪 3

concord[1] ---------------------------> BATS 🡪 1

The pattern continues.

**concordanceBST.h – FIRST HEADER**

#include <iostream>

#include <new>

#include <fstream>

#include <string>

#include <sstream>

#include "token.h"

using namespace std;

#ifndef BINARY\_SEARCH\_TREE

#define BINARY\_SEARCH\_TREE

template <typename DataType>

class BST

{

public:

/\*\*\*\*\* Function Members \*\*\*\*\*/

BST();

/\*------------------------------------------------------------------------

Construct a BST object.

Precondition: None.

Postcondition: An empty BST has been constructed.

-----------------------------------------------------------------------\*/

bool empty() const;

/\*------------------------------------------------------------------------

Check if BST is empty.

Precondition: None.

Postcondition: Returns true if BST is empty and false otherwise.

-----------------------------------------------------------------------\*/

bool search(const DataType & item) const;

/\*------------------------------------------------------------------------

Search the BST for item.

Precondition: None.

Postcondition: Returns true if item found, and false otherwise.

-----------------------------------------------------------------------\*/

void insert(DataType & item);

/\*------------------------------------------------------------------------

Insert item into BST.

Precondition: None.

Postcondition: BST has been modified with item inserted at proper

position to maintain BST property.

------------------------------------------------------------------------\*/

void remove(const DataType & item);

/\*------------------------------------------------------------------------

Remove item from BST.

Precondition: None.

Postcondition: BST has been modified with item removed (if present);

BST property is maintained.

Note: remove uses private auxiliary function search2() to locate

the node containing item and its parent.

------------------------------------------------------------------------\*/

void graph(ostream & out) const;

/\*------------------------------------------------------------------------

Graphic output of BST.

Precondition: ostream out is open.

Postcondition: Graphical representation of BST has been output to out.

Note: graph() uses private auxiliary function graphAux().

------------------------------------------------------------------------\*/

void inorder(ostream & out) const;

/\*------------------------------------------------------------------------

Inorder traversal of BST.

Precondition: ostream out is open.

Postcondition: BST has been inorder traversed and values in nodes

have been output to out.

Note: inorder uses private auxiliary function inorderAux().

------------------------------------------------------------------------\*/

//--- ADD PROTOTYPES OF preorder() AND postorder() HERE

void preorder(ostream & out) const;

void postorder(ostream & out) const;

//--- ADD PROTOTYPE OF DESTRUCTOR HERE

~BST()

{

destroy(myRoot);

}

//--- ADD PROTOTYPE OF COPY CONSTRUCTOR HERE

BST(const BST<DataType> &original)

{

copyTree(original.myRoot, myRoot);

}

//--- ADD PROTOTYPE OF ASSIGNMENT OPERATOR HERE

BST<DataType>& operator=(const BST<DataType>& origList);

private:

/\*\*\*\*\* Node class \*\*\*\*\*/

class BinNode

{

public:

DataType data;

BinNode \* left;

BinNode \* right;

// BinNode constructors

// Default -- data part is default DataType value; both links are null.

BinNode()

: left(0), right(0)

{}

// Explicit Value -- data part contains item; both links are null.

BinNode(DataType item)

: data(item), left(0), right(0)

{}

};// end of class BinNode declaration

typedef BinNode \* BinNodePointer;

/\*\*\*\*\* Private Function Members \*\*\*\*\*/

void search2(const DataType & item, bool & found,

BinNodePointer & locptr, BinNodePointer & parent) const;

/\*------------------------------------------------------------------------

Locate a node containing item and its parent.

Precondition: None.

Postcondition: locptr points to node containing item or is null if

not found, and parent points to its parent.#include <iostream>

------------------------------------------------------------------------\*/

void destroy(BinNodePointer subtreeRoot)

{

if (subtreeRoot != NULL)

{

destroy(subtreeRoot->left);

destroy(subtreeRoot->right);

delete subtreeRoot;

}

}

void copyTree(BinNodePointer origRoot, BinNodePointer &subtreeRoot)

{

if (origRoot == NULL)

{

subtreeRoot = NULL;

}

else

{

subtreeRoot = new BinNode(origRoot->data);

copyTree(origRoot->left, subtreeRoot->left);

copyTree(origRoot->right, subtreeRoot->right);

}

}

void inorderAux(ostream & out,

BinNodePointer subtreePtr) const;

void preorderAux(ostream & out,

BinNodePointer subtreePtr) const;

void postorderAux(ostream & out,

BinNodePointer subtreePtr) const;

/\*------------------------------------------------------------------------

Inorder traversal auxiliary function.

Precondition: ostream out is open; subtreePtr points to a subtree

of this BST.

Postcondition: Subtree with root pointed to by subtreePtr has been

output to out.

------------------------------------------------------------------------\*/

void graphAux(ostream & out, int indent,

BinNodePointer subtreeRoot) const;

/\*------------------------------------------------------------------------

Graph auxiliary function.

Precondition: ostream out is open; subtreePtr points to a subtree

of this BST.

Postcondition: Graphical representation of subtree with root pointed

to by subtreePtr has been output to out, indented indent spaces.

------------------------------------------------------------------------\*/

/\*\*\*\*\* Data Members \*\*\*\*\*/

BinNodePointer myRoot;

}; // end of class template declaration

//--- Definition of constructor

template <typename DataType>

inline BST<DataType>::BST()

: myRoot(0)

{}

//--- Definition of empty()

template <typename DataType>

inline bool BST<DataType>::empty() const

{

return myRoot == 0;

}

//--- Definition of search()

template <typename DataType>

bool BST<DataType>::search(const DataType & item) const

{

BST<DataType>::BinNodePointer locptr = myRoot;

bool found = false;

while (!found && locptr != 0)

{

if (item < locptr->data) // descend left

locptr = locptr->left;

else if (locptr->data < item) // descend right

locptr = locptr->right;

else // item found

found = true;

}

return found;

}

//--- Definition of insert()

template <typename DataType>

inline void BST<DataType>::insert(DataType & item)

{

BST<DataType>::BinNodePointer

locptr = myRoot, // search pointer

parent = 0; // pointer to parent of current node

bool found = false; // indicates if item already in BST

while (!found && locptr != 0)

{

parent = locptr;

if (item < locptr->data) // descend left

locptr = locptr->left;

else if (locptr->data < item) // descend right

locptr = locptr->right;

else // item found

found = true;

}

if (!found)

{ // construct node containing item

locptr = new(nothrow) BST<DataType>::BinNode(item);

if (locptr == 0)

{

cerr << "\*\*\* Out of memory -- terminating program \*\*\*\n";

exit(1);

}

if (parent == 0) // empty tree

myRoot = locptr;

else if (item < parent->data) // insert to left of parent

parent->left = locptr;

else // insert to right of parent

parent->right = locptr;

}

else if (found)

{

locptr->data.a.enqueue(item.lineNum);

}

}

//--- Definition of remove()

template <typename DataType>

void BST<DataType>::remove(const DataType & item)

{

bool found; // signals if item is found

BST<DataType>::BinNodePointer

x, // points to node to be deleted

parent; // " " parent of x and xSucc

search2(item, found, x, parent);

if (!found)

{

cout << "Item not in the BST\n";

return;

}

//else

if (x->left != 0 && x->right != 0)

{ // node has 2 children

// Find x's inorder successor and its parent

BST<DataType>::BinNodePointer xSucc = x->right;

parent = x;

while (xSucc->left != 0) // descend left

{

parent = xSucc;

xSucc = xSucc->left;

}

// Move contents of xSucc to x and change x

// to point to successor, which will be removed.

x->data = xSucc->data;

x = xSucc;

} // end if node has 2 children

// Now proceed with case where node has 0 or 2 child

BST<DataType>::BinNodePointer

subtree = x->left; // pointer to a subtree of x

if (subtree == 0)

subtree = x->right;

if (parent == 0) // root being removed

myRoot = subtree;

else if (parent->left == x) // left child of parent

parent->left = subtree;

else // right child of parent

parent->right = subtree;

delete x;

}

//--- Definition of graph()

template <typename DataType>

inline void BST<DataType>::graph(ostream & out) const

{

graphAux(out, 0, myRoot);

}

//--- Definition of search2()

template <typename DataType>

void BST<DataType>::search2(const DataType & item, bool & found,

BinNodePointer & locptr,

BinNodePointer & parent) const

{

locptr = myRoot;

parent = 0;

found = false;

while (!found && locptr != 0)

{

if (item < locptr->data) // descend left

{

parent = locptr;

locptr = locptr->left;

}

else if (locptr->data < item) // descend right

{

parent = locptr;

locptr = locptr->right;

}

else // item found

found = true;

}

}

//--- Definition of graphAux()

#include <iomanip>

template <typename DataType>

void BST<DataType>::graphAux(ostream & out, int indent,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

graphAux(out, indent + 8, subtreeRoot->right);

out << setw(indent) << " " << subtreeRoot->data << endl;

graphAux(out, indent + 8, subtreeRoot->left);

}

}

//--- Definition of inorder()

template <typename DataType>

inline void BST<DataType>::inorder(ostream & out) const

{

inorderAux(out, myRoot);

}

//--- Definition of inorderAux()

template <typename DataType>

void BST<DataType>::inorderAux(ostream & out,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

inorderAux(out, subtreeRoot->left); // L operation

out << subtreeRoot->data << " "; // V operation

inorderAux(out, subtreeRoot->right); // R operation

}

}

//--- PUT DEFINITIONS OF THE ADDED OPERATIONS HERE

//assignment operator

template <typename DataType>

BST<DataType>& BST<DataType>::operator=(const BST<DataType>& origRoot)

{

if (this != &origRoot)

{

destroy(myRoot);

copyTree(origRoot.myRoot, myRoot);

}

return \*this;

}

//--- Definition of preorder()

template <typename DataType>

inline void BST<DataType>::preorder(ostream & out) const

{

preorderAux(out, myRoot);

}

//--- Definition of preorderAux()

template <typename DataType>

void BST<DataType>::preorderAux(ostream & out,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

out << subtreeRoot->data << " "; // V operation

inorderAux(out, subtreeRoot->left); // L operation

inorderAux(out, subtreeRoot->right); // R operation

}

}

template <typename DataType>

inline void BST<DataType>::postorder(ostream & out) const

{

postorderAux(out, myRoot);

}

//--- Definition of postorderAux()

template <typename DataType>

void BST<DataType>::postorderAux(ostream & out,

BinNodePointer subtreeRoot) const

{

if (subtreeRoot != 0)

{

inorderAux(out, subtreeRoot->left); // L operation

inorderAux(out, subtreeRoot->right); // R operation

out << subtreeRoot->data << " "; // V operation

}

}

#endif

**Token.h – SECOND HEADER**

#include <iostream>

#include <new>

#include <fstream>

#include <string>

#include "Queue.h"

#include "Queue.cpp"

using namespace std;

class token

{

public:

int lineNum; //line number

string word; //the word

Queue a; //the queue of word currenences

//display function

//Precons: ostream

//Postcons: none

void display(ostream &out) const

{

cout << word << ", line numbers: " << a << endl;

}

};

//overloaded output operator

ostream &operator<<(ostream& out, const token &q)

{

q.display(out);

return out;

}

//overloaded less than operator

inline bool operator <(const token& lhs, const token &rhs)

{

return lhs.word < rhs.word;

}

**Queue.h – THIRD HEADER**

#ifndef QUEUE

#define QUEUE

#include<string>

#include<iostream>

#include<fstream>

using namespace std;

const int QUEUE\_CAPACITY = 128;

typedef int QueueElement;

class Queue

{

public:

/\*\*\*\*\* Function Members \*\*\*\*\*/

/\*\*\*\*\* Constructor \*\*\*\*\*/

Queue();

/\*-----------------------------------------------------------------------

Construct a Queue object.

Precondition: None.

Postcondition: An empty Queue object has been constructed; myFront

and myBack are initialized to -1 and myArray is an array with

QUEUE\_CAPACITY elements of type QueueElement.

----------------------------------------------------------------------\*/

/\*\*\*\*\* Copy \*\*\*\*\*/

Queue(const Queue & original);

/\*\*\*\*\* Destructor \*\*\*\*\*/

~Queue();

/\*------------------------------------------------------------------------

Class destructor

Precondition: None

Postcondition: The linked list in the stack has been deallocated.

------------------------------------------------------------------------\*/

/\*\*\*\*\* Assignment \*\*\*\*\*/

const Queue & operator= (const Queue & rightHandSide);

/\*------------------------------------------------------------------------

Assignment Operator

Precondition: rightHandSide is the stack to be assigned and is

received as a const reference parameter.

Postcondition: The current stack becomes a copy of rightHandSide

and a const reference to it is returned.

------------------------------------------------------------------------\*/

bool empty() const;

/\*-----------------------------------------------------------------------

Check if queue is empty.

Precondition: None.

Postcondition: True is returned if the queue is empty and false is

returned otherwise.

----------------------------------------------------------------------\*/

void enqueue(const QueueElement & value);

/\*-----------------------------------------------------------------------

Add a value to a queue.

Precondition: value is to be added to this queue.

Postcondition: value is added to back of queue provided there is space;

otherwise, a queue-full message is displayed and execution is

terminated.

-----------------------------------------------------------------------\*/

void display(ostream & out) const;

/\*-----------------------------------------------------------------------

Output the values stored in the queue.

Precondition: ostream out is open.

Postcondition: Queue's contents, from front to back, have been output

to out.

-----------------------------------------------------------------------\*/

QueueElement front() const;

/\*-----------------------------------------------------------------------

Retrieve value at front of queue (if any).

Precondition: Queue is nonempty.

Postcondition: Value at front of queue is returned, unless queue is

empty; in that case, an error message is displayed and a "garbage

value" is returned.

----------------------------------------------------------------------\*/

void dequeue();

/\*-----------------------------------------------------------------------

Remove value at front of queue (if any).

Precondition: Queue is nonempty.

Postcondition: Value at front of queue has been removed, unless queue

is empty; in that case, an error message is displayed and

execution is terminated.

----------------------------------------------------------------------\*/

private:

/\*\*\*\*\* Data Members \*\*\*\*\*/

int myFront,

myBack;

int myCapacity;

QueueElement \*myArray = new QueueElement[myCapacity];

}; // end of class declaration

//overloaded output operator

ostream &operator<<(ostream &out, const Queue &q);

#endif

**Queue.cpp – QUEUE.H IMPLEMENTATION**

#include<string>

#include<iostream>

#include<fstream>

using namespace std;

//default constructor

Queue::Queue()

{

myCapacity = 25;

myArray = new(nothrow)QueueElement[myCapacity];

if (myArray != 0)

{

myFront = 0;

myBack = 0;

}

else

{

cout << "Insufficient Space" << endl;

exit(1107);

}

}

//copy constructor

//Precons: original Queue

//Postcons: lnewly copied myArray

Queue::Queue(const Queue & original)

: myCapacity(original.myCapacity), myFront(original.myFront),myBack(original.myBack)

{

myArray = new(nothrow) QueueElement[myCapacity];

if (myArray != 0)

{

for (int i = myFront; i != myBack; i = (i + 1) % myCapacity)

myArray[i] = original.myArray[i];

}

else

{

cout << "Insufficient Memory" << endl;

exit(1107);

}

}

//destructor

Queue::~Queue()

{

delete[] myArray;

}

const Queue & Queue::operator= (const Queue & rightHandSide)

{

if (this != &rightHandSide)

{

delete[] myArray;

myCapacity = rightHandSide.myCapacity;

myArray = new QueueElement[myCapacity];

if (myArray == 0)

{

cout << "Insufficient Space" << endl;

exit(1107);

}

}

myFront = rightHandSide.myFront;

myBack = rightHandSide.myBack;

for (int i = myFront; i != myBack; i = (i + 1) % myCapacity)

{

myArray[i] = rightHandSide.myArray[i];

}

return \*this;

}

//Empty function, returns if the queue is empty

//Precons: none

//Postcons: bool

bool Queue::empty() const

{

return (myFront == myBack);

}

//Enqueue function, adds a value to the queue

//Precons: a value to enqueue

//Postcons: none

void Queue::enqueue(const QueueElement & value)

{

int newBack = (myBack + 1) % QUEUE\_CAPACITY;

if (newBack != myFront) // queue isn't full

{

myArray[myBack] = value;

myBack = newBack;

}

else

{

cerr << "\*\*\* Queue full -- can't add new value \*\*\*\n"

"Must increase value of QUEUE\_CAPACITY in Queue.h\n";

exit(1);

}

}

//Display function, displays current queue

//Precons: outstream

//Postcons: none

void Queue::display(ostream & out) const

{

for (int i = myFront; i != myBack; i = (i + 1)%QUEUE\_CAPACITY)

out << myArray[i] << " ";

cout << endl;

}

//overloaded output operator

//Precons: outstream, Queue to output

//Postcons: outstream

ostream &operator<<(ostream &out, const Queue &q)

{

q.display(out);

return out;

}

//front function, returns value at the front of the queue

//Precons: none

//Postcons: front value

QueueElement Queue::front() const

{

if ( !empty() )

return (myArray[myFront]);

else

{

cerr << "\*\*\* Queue is empty -- returning garbage value \*\*\*\n";

QueueElement garbage = 0;

return garbage;

}

}

//dequeue function, moves myFront ptr forward

//Precons: none

//Postcons: none

void Queue::dequeue()

{

if ( !empty() )

myFront = (myFront + 1) % QUEUE\_CAPACITY;

else

{

cerr << "\*\*\* Queue is empty -- "

"can't remove a value \*\*\*\n";

exit(1);

}

}

**ConcordanceDriver.cpp – DRIVER**

#include <iostream>

#include <new>

#include <fstream>

#include <string>

#include <sstream>

#include "concordanceBST.h"

using namespace std;

int main()

{

BST<token> b[26]; //BST of tokens

int lineNumber = 1; //line number count

ifstream in("text.txt"); //text file to sort

string line; //current word passed in

char c; //line number variable

if (!in) //check file status

{

cout << "error opening file";

exit(1107);

}

while (in >> line)

{

//capitalize all words

for (int i = 0; i < line.length(); i++)

{

line[i] = toupper(line[i]);

}

//remove all punctuation

for (int i = 0, len = line.size(); i < len; i++)

{

if (ispunct(line[i]))

{

line.erase(i--, 1);

len = line.size();

}

}

//count the line number

c = in.get();

if (c == '\n')

{

lineNumber++;

}

//start inserting into the BST

token temp;

temp.lineNum = lineNumber;

temp.word = line;

temp.a.enqueue(lineNumber);

b[line[0] - 'A'].insert(temp);

}

//print the BST

for (int i = 0; i < 26; i++)

{

cout << i << ") ";

b[i].inorder(cout);

cout << endl;

}

}

**TEXT.TXT:**

**Ants and bats**

**and cows and cats**

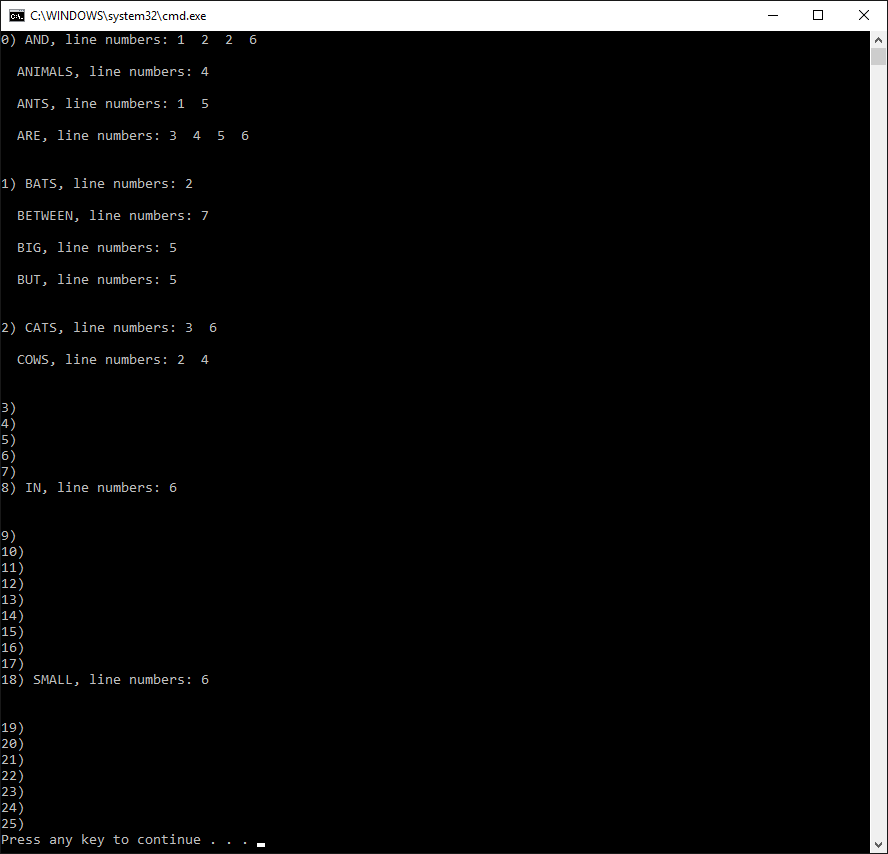
**are animals**

**Cows are big**

**but ants are small**

**and cats are in**

**between**



**TEXT.TXT:**

**This is just some sample text**

**that appears on several lines. This**

**is a test of the Concordance program**

**to see how well it is able to pick out**

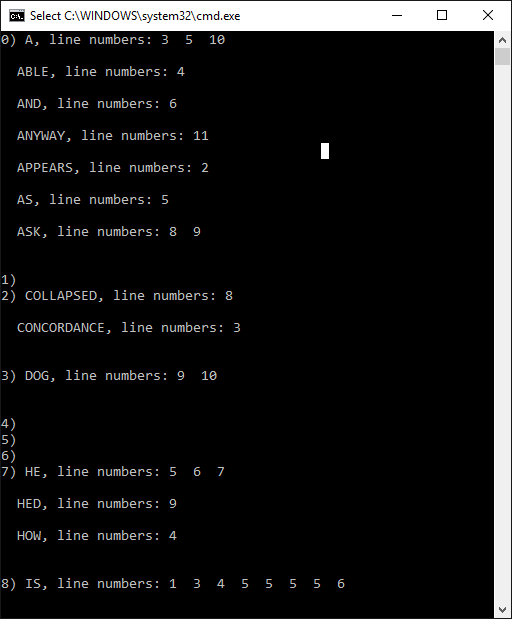
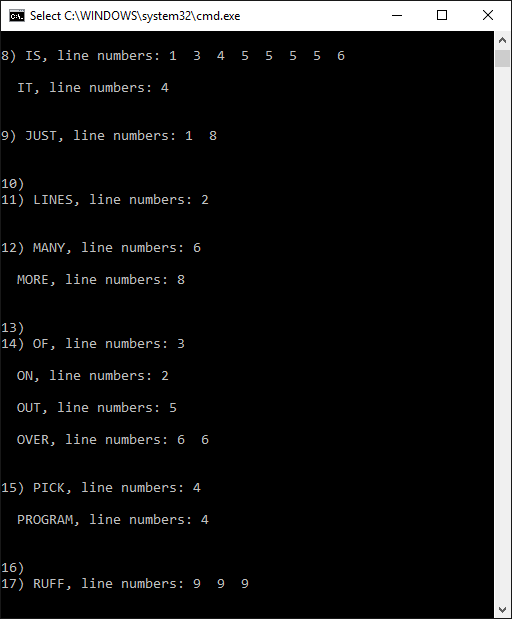
**the words. As a test, he said: "is is is is is,"**

**many times, over and over, until he was so**

**tired he collapsed.**

**There's more to this story; just ask the dog.**

**He'd say: "Ruff, Ruff, Ruff!" Who'd ask a**

**dog, anyway?**