```
/* sets.cpp
 *
 * CS 121.Bolden.........GCC 11.4.0...........Jake Gendreau
 * Feb 16, 2024 ................Pop!_OS 22.04 / Core i9-13900H.............gend0188@vandal
s.uidaho.edu
 *
 * Taking in two data sets, output the intersection and the union between the two.
 *-------------------------------------------------------------------------
 */

#include <iostream>
#include <fstream>

using namespace std;

struct node{
    node* next;
    string data;
};

typedef struct node* NodePtr;

//prototypes
void readFile(NodePtr&, string);
void appendToList(string, NodePtr&);
void printList(NodePtr);
void subtractIntersect(NodePtr&, NodePtr&);
void deleteNode(string, NodePtr&);

NodePtr findIntersect(NodePtr&, NodePtr&);
NodePtr concatLists(NodePtr&, NodePtr&);

bool searchList(string, NodePtr&);
bool isAlpha(char);

string cleanWord(string);

int main(int argc, char* argv[]){
    if(argc != 3){
        cout << "Incorrect usage: ./a.out <set 1> <set 2>" << endl;
        return 1;
    }

    //define list heads
    NodePtr d1Head = NULL;
    NodePtr d2Head = NULL;

    //read in data from files
    readFile(d1Head, argv[1]);
    readFile(d2Head, argv[2]);

    //find intersect and union lists
    NodePtr intersect = findIntersect(d1Head, d2Head);
    NodePtr setUnion = concatLists(d1Head, d2Head);
    subtractIntersect(intersect, setUnion);


    //print intersect and union
    cout << "Intersection:" << endl;
    printList(intersect);
    cout << endl;

    cout << "Union:" << endl;
    printList(setUnion);
    cout << endl;


}
```

```
void subtractIntersect(NodePtr &intersect, NodePtr &setUnion){
    //start at beginning of list
    NodePtr p = intersect;
    string query;

    //iterate through each element of the intersect, deleting from the union
    while(p != NULL){
        query = p -> data;
        deleteNode(query, setUnion);

        p = p -> next;
    }
}

NodePtr findIntersect(NodePtr &d1Head, NodePtr &d2Head){
    //define new node pointers
    NodePtr intersect = NULL;
    NodePtr p = d1Head;

    string query;

    //iterate through every element of d1
    while(p != NULL){
        //set new search query
        query = p -> data;

        //append if query is in the 2nd list
        if(searchList(query, d2Head)){
            appendToList(query, intersect);
        }

        p = p -> next;
    }

    return intersect;
}

NodePtr concatLists(NodePtr& d1Head, NodePtr& d2Head){
    NodePtr setUnion = NULL;
    NodePtr p = d2Head;

    //copy elements to setUnion
    while(p != NULL){
        appendToList(p -> data, setUnion);

        p = p -> next;
    }

    p = d1Head;

    //copy elements to setUnion
    while(p != NULL){
        appendToList(p -> data, setUnion);

        p = p -> next;
    }

    return setUnion;
}

void readFile(NodePtr &head, string filename){
    string word;
    fstream file;

    file.open(filename);

    //append if word has not already been seen
```

```
    while(file >> word){
        word = cleanWord(word);
        if(!searchList(word, head)){
            appendToList(word, head);
        }
    }

    file.close();
}

string cleanWord(string inString){
    string buffer;

    //write characters to buffer, ignoring punctuation
    for(int i = 0; inString[i] != '\0'; i++){
        if(isAlpha(inString[i])){
            buffer += (inString[i]);
        }
    }

    return buffer;
}

bool isAlpha(char curChar){ //isAlpha with extras!
    if((curChar >= 'a' && curChar <= 'z') || (curChar >= 'A' && curChar <= 'Z') || curChar
== '-' || curChar == '\'')
        return true;

    return false;
}

void appendToList(string newData, NodePtr &head){
    //make new node
    NodePtr p = new node;

    //set node data
    p -> data = newData;
    p -> next = NULL;

    //if head is the only element, head = the new node
    if(head == NULL){
        head = p;
    }

    //otherwise, add p before head, and make head = p
    else {
        p -> next = head;
        head = p;
    }
}

void deleteNode(string query, NodePtr &head){ //next doesn't feel like a word anymore
    //start at beginning of list
    NodePtr p = head;

    //test first element
    if(p -> data == query){
        //skip current node
        head = p -> next;

        //delete old node
        p -> next = NULL;
        delete(p);
        return;
    }

    //iterate through, testing for element
    while(p != NULL){
```

```
        NodePtr nextNode = p -> next;

        //exit at end of list
        if(nextNode == NULL){
            return;
        }

        if(nextNode -> data == query){
            //update next to skip next node
            p -> next = nextNode -> next;

            //delete old node
            nextNode -> next = NULL;
            delete(nextNode);
            return;
        }

        p = p -> next;
    }
}

bool searchList(string query, NodePtr &head){
    //start at beginning of list
    NodePtr p = head;

    //iterate through, testing for element
    while(p != NULL){
        if(p -> data == query)
            return true;

        p = p -> next;
    }

    return false;
}

void printList(NodePtr head){
    //start at beginning of list
    NodePtr p = head;

    //iterate through, printing every element along the way
    while(p != NULL){
        cout << p -> data << endl;
        p = p -> next;
    }
}
```