```cpp
//q1
#include <iostream>
#include <sstream>
#include <vector>
#include <string>
#include <algorithm>
#include <cctype>

// Function to count words in a line
int countWords(const std::string &line) {
    std::istringstream iss(line);
    int count = 0;
    std::string word;
    while (iss >> word) {
        count++;
    }
    return count;
}

int main() {
    std::vector<std::pair<int, std::string>> lines;
    std::string input;
    int line_number = 0;

    // Reading input lines until EOF
    while (std::getline(std::cin, input)) {
        int wordCount = countWords(input);
        lines.emplace_back(wordCount, input);
        line_number++;
    }

    // Sorting lines by word count and initial order
    std::stable_sort(lines.begin(), lines.end(), [](const auto &a, const auto &b) {
        return a.first < b.first;
    });

    // Output results
    for (const auto &entry : lines) {
        std::cout << entry.first << ": " << entry.second << "\n";
    }

    return 0;
}


// q2 linkedlist
void SingleLink::SwapFirst2() {
    if (!head_ || !head_->next_) return;  // Not enough elements to swap
    Node *temp = head_->next_; // temporary node to store the second node
    head_->next_ = temp->next_; // first node now points to the third node
    temp->next_ = head_; // second node now points to the first node
    head_ = temp; // second node is now the head
}

void SingleLink::SwapLast2() {
    // If the list has less than 2 elements, swapping last two is not possible
    if (head_ == nullptr || head_->next_ == nullptr) {
        return;
    }

    // Find the second to last node and the last node
    Node* prev = nullptr;
    Node* curr = head_;
    while (curr->next_ != nullptr) {
        prev = curr;
        curr = curr->next_;
    }

    // At this point, prev is the second to last node, and curr is the last node
    if (prev != nullptr) {
```

```
            std::swap(prev->data_, curr->data_);
        }
    }

//q3   alphabet position
#include <iostream>
#include <string>

int AlphabetPosition(char letter) {
    if (letter >= 'a' && letter <= 'z') {
        return letter - 'a';
    } else if (letter >= 'A' && letter <= 'Z') {
        return letter - 'A';
    } else {
        return -1;
    }
}

int main() {
    char letter;
    std::cin >> letter;
    std::cout << AlphabetPosition(letter) << std::endl;
    return 0;
}

// overunder question
// header.hpp
#pragma once

#include <list>
#include <iostream>

class OverUnder {
private:
    std::list<int> guesses_; // List to store guesses

public:
    OverUnder(); // Constructor
    void add(int a); // Adds a guess to the list
    int over(int a) const; // Returns count of elements greater than 'a'
    int under(int a) const; // Returns count of elements less than 'a'
};

// implementation.cpp
#include "header.hpp"
#include <iostream>

OverUnder::OverUnder() {
    guesses_ = std::list<int>();
}

void OverUnder::add(int a) {
    guesses_.push_back(a);
}

int OverUnder::over(int a) const {
    int count = 0;
    for (int i : guesses_) {
        if (i > a) {
            count++;
        }
    }
    return count;
}

int OverUnder::under(int a) const {
    int count = 0;
    for (int i : guesses_) {
        if (i < a) {
            count++;
```

```
        }
    }
    return count;
}
// main.cpp
#include "header.hpp"
#include <iostream>

int main() {
    OverUnder game;
    int guess;
    bool isFirstGuess = true;
    int referenceValue = 0; // To store the first input as the reference value

    while (true) {
        std::cin >> guess;

        if (guess == -1) {
            // Output the number of guesses in the list
            std::cout << "Number of guesses: " << game.over(-1000000) + game.under(1000000) << std::endl;
        } else if (guess == 0) { // Assume 0 is the correct number to stop
            break;
        } else {
            if (isFirstGuess) {
                // Set the first guess as the reference value
                referenceValue = guess;
                isFirstGuess = false;
            }

            game.add(guess);
            // Compare subsequent inputs to the first input (reference value)
            std::cout << game.over(referenceValue) << " " << game.under(referenceValue) << std::endl;
        }
    }

    return 0;
}


// phonebook question
// PhoneBook.hpp
#pragma once

#include <string>
#include <vector>
#include <iostream>

struct Entry {
    std::string name_;
    std::string phone_number_;
};

class PhoneBook {
    private:
        std::vector<Entry> entries_;

    public:
        PhoneBook();
        void AddEntry(const std::string &name, const std::string &phone_number);
        void RemoveEntry(const std::string &phone_number);
        void Display() const;

        // what I add
        void OnlyLocalEntries();
        int size() const { return entries_.size(); }

        friend std::ostream& operator<<(std::ostream &out, const PhoneBook &pb);
};

// implementation.cpp
```

```cpp
#include "header.h"

#include <iostream>

PhoneBook::PhoneBook() {}

void PhoneBook::AddEntry(const std::string &name, const std::string &phone_number) {
    Entry newEntry;
    newEntry.name_ = name;
    newEntry.phone_number_ = phone_number;
    entries_.push_back(newEntry);
}

void PhoneBook::RemoveEntry(const std::string &phone_number) {
    for (size_t i = 0; i < entries_.size(); i++) {
        if (entries_[i].phone_number_ == phone_number) {
            entries_.erase(entries_.begin() + i);
            return;
        }
    }
}

void PhoneBook::Display() const {
    for (size_t i = 0; i < entries_.size(); i++) {
        std::cout << entries_[i].name_ << " " << entries_[i].phone_number_ << std::endl;
    }
}

std::ostream& operator<<(std::ostream &out, const PhoneBook &pb) {
    for (size_t i = 0; i < pb.entries_.size(); i++) {
        out << pb.entries_[i].name_ << " " << pb.entries_[i].phone_number_ << std::endl;
    }
    return out;
}

// what I add
void PhoneBook::OnlyLocalEntries() {
    for (size_t i = 0; i < entries_.size(); i++) {
        if (entries_[i].phone_number_.size() != 12) {
            entries_.erase(entries_.begin() + i);
            i--;
        } else if (entries_[i].phone_number_.substr(0, 3) != "517") {
            entries_.erase(entries_.begin() + i);
            i--;
        }
    }
}

// main.cpp
#include <iostream>
#include <string>
#include <vector>
#include <cassert>
#include "header.h"

int main() {
    PhoneBook pb;
    pb.AddEntry("John", "517-123-4567");
    pb.AddEntry("Jane", "517-123-4567");
    pb.AddEntry("Jake", "517-123-4567");
    pb.AddEntry("Jill", "424-123-4567");
    pb.AddEntry("James", "081-123-4567");

    pb.OnlyLocalEntries();

    //Assertions
    // assert(pb.entries_.size() == 3);
    // assert(pb.entries_[0].name_ == "John");
    // assert(pb.entries_[1].name_ == "Jane");
```

```
    // assert(pb.entries_[2].name_ == "Jake");

    //Output the local entries for verification
    pb.Display();

    return 0;
}
```