```
Script started on 2024-09-15 13:11:43-05:00 [TERM="xterm-256color" TTY="/dev/pts/3'
e_prevost@ares:~/Portfolio_1/Lab 2$ pwd
/home/students/e_prevost/Portfolio_1/Lab 2
e_prevost@ares:~/Portfolio_1/Lab 2$ cat seqlist.info
****************

Robert Prevost

CSC 122 W01



Seq List Lab



Takes in a problem set list and outputs the problems

required with the problem set name.



Base Level: Level 4

Total Level: Level 4

****************e_prevost@ares:~/Portfolio_1/Lab 2$ show-code driver_2.cpp


driver_2.cpp:


#include <iostream>
#include <string>
#include "seqlist.h"
#include "input_prot.h"
using namespace std;

int main() {
    string psn = "Please Input your problem set's name: ";
    string ps = "Please Input your problem set: ";
    string pta = "Please Try Again.";
    string programName = check_input<string>(psn, pta);
    string numberline = check_input<string>(ps,pta);
    string output_line = convertLineToString(programName,numberline);
    cout << output_line;
}
e_prevost@ares:~/Portfolio_1/Lab 2$ showcode seqlist.h
showcode: command not found
e_prevost@ares:~/Portfolio_1/Lab 2$ show-code seqlist.h


seqlist.h:


#ifndef seqlist_h
#define seqlist_h
#include <string>
#include <vector>

std::string getStringFromTwoNumbers(char firstnum, char secondnum);
std::string convertLineToString(std::string lesson_name, std::string
problem_number_line);
std::vector<std::string> splitProblemNumbers(std::string
problem_number_line);
std::vector<std::string> sortAndUnique(std::vector<std::string> vec);
void customBubbleSort(std::vector<std::string>& vec);

#endif
e_prevost@ares:~/Portfolio_1/Lab 2$ show-code seqlist.cpp


seqlist.cpp:


#include <iostream>
#include <string>
#include <ctype.h>
#include <vector>
#include "seqlist.h"
using namespace std;

void customBubbleSort(vector<string>& vec) {
    int n = vec.size();
    bool swapped;

    for (int i = 0; i < n - 1; i++) {
        swapped = false;

        for (int j = 0; j < n - i - 1; j++) {
            int num1 = stoi(vec[j]);
            int num2 = stoi(vec[j + 1]);

            if (num1 > num2) {
                swap(vec[j], vec[j + 1]);
                swapped = true;
            }
        }

        if (!swapped) {
            break;
        }
    }
}

vector<string> sortAndUnique(vector<string> vec) {
    customBubbleSort(vec);

```

```cpp
    vector<string> result;
    for (string s : vec) {
        if (result.empty() || s != result.back()) {
            result.push_back(s);
        }
    }

    return result;
}

vector<string> getStringFromTwoNumbers(string firstnum, string secondnum)

    int first = stoi(firstnum);
    int second = stoi(secondnum);

    vector<string> numStrings;

    for (int i = first; i <= second; i++) {
        numStrings.push_back(to_string(i));
    }

    return numStrings;
}


vector<string> splitProblemNumbers(string problem_number_line) {
    vector<string> result;
    string current_number;

    for (char c : problem_number_line) {
        if (isdigit(c)) {
            current_number += c;
        } else if (c == ',' || c == '-') {
            if (!current_number.empty()) {
                result.push_back(current_number);
                current_number.clear();
            }
            result.push_back(string(1, c));
        }
    }

    if (!current_number.empty()) {
        result.push_back(current_number);
    }

    return result;
}

string convertLineToString(string lesson_name, string problem_number_line)
    string lesson_name_clrd;
    for(char c : lesson_name){
        if(c != '"' && c != '\''){
            lesson_name_clrd += c;
        }
    }
    vector<string> number_string;
    vector<string> problem_number_vector = splitProblemNumbers
    (problem_number_line);
    for(size_t i = 0; i < problem_number_vector.size(); i++){
        if(problem_number_vector[i] != string("-") || problem_number_vecto
        [i] != string(",")){
            if(i + 1 < problem_number_vector.size() &&
            problem_number_vector[i+1][0] == ','){
                number_string.push_back(problem_number_vector[i]);
                i++;
            }
            else if(i + 2 < problem_number_vector.size() &&
            problem_number_vector[i+1][0] == '-')
            {
                vector<string> number_addition = getStringFromTwoNumbers
                (problem_number_vector[i],problem_number_vector[i+2]);

                for(string b: number_addition){
                    if(!b.empty() && b.find(',') == -1){
                       number_string.push_back(b);
                    }
                }
                i+=2;
            }
            else if(!problem_number_vector[i].empty() &&
            !problem_number_vector[i].find(',')  == -1){
                number_string.push_back(problem_number_vector[i]);
            }
        }
    }

    vector<string> new_number_string = sortAndUnique(number_string);

    string output_string = "Do Problem";
    if(new_number_string.size() > 1){
        output_string += "s";
    }
    output_string += " ";

    for(size_t i = 0; i < new_number_string.size(); i++){
        if(i > 0) {
            if(i == new_number_string.size() - 1){
                output_string += " and ";
            } else {
                output_string += ", ";
            }
        }
        output_string += new_number_string[i];
    }

    output_string += " of " + lesson_name_clrd;
    return output_string;

```

```
}
e_prevost@ares:~/Portfolio_1/Lab 2$ show-code input_prot.h


input_prot.h:


#ifndef input_protection_h
#define input_protection_h
#include <string>
#include <iostream>
#include <limits>
#include <vector>
template <typename varType>
varType check_input(std::string first_prompt, std::string try_again)
{
    std::cout << first_prompt;
    varType input;
    std::cin >> input;
    while (std::cin.fail()){
        std::cerr << try_again;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n')
        std::cin >> input;
    }
    return input;
}


template <typename varType>
varType check_input(varType min,std::string first_prompt, std::string
try_again, varType max )
{
    std::cout << first_prompt;
    bool passed_min_max = false;
    varType input;
    std::cin >> input;
    if(input > min && input < max && !std::cin.fail()){
        passed_min_max = true;
    }
    while (std::cin.fail() || !passed_min_max){
        std::cerr << try_again;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n')
        std::cin >> input;
        if(input > min && input < max && !std::cin.fail()){
            passed_min_max = true;
        }
    }
    return input;
}

template <typename varType>
varType check_input(varType min, std::string first_prompt, std::string
try_again)
{
    std::cout << first_prompt;
    bool passed_min_max = false;
    varType input;
    std::cin >> input;
    if(input > min && !std::cin.fail()){
        passed_min_max = true;
    }
    while (std::cin.fail() || !passed_min_max){
        std::cerr << try_again;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n')
        std::cin >> input;
        if(input > min && !std::cin.fail()){
            passed_min_max = true;
        }
    }
    return input;
}


template <typename varType>
varType check_input( std::string first_prompt, std::string try_again,
varType max)
{
    std::cout << first_prompt;
    bool passed_min_max = false;
    varType input;
    std::cin >> input;
    if(input < max && !std::cin.fail()){
        passed_min_max = true;
    }
    while (std::cin.fail() || !passed_min_max){
        std::cerr << try_again;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n')
        std::cin >> input;
        if(input < max && !std::cin.fail()){
            passed_min_max = true;
        }
    }
    return input;
}

template <typename varType>
bool isValueInArray(const varType value, const std::vector<std::string>
arr)
{
    std::string value_str;
    if (std::is_same<varType, char>::value) {
        value_str = std::string(1, value); //when char gets converted to
        //string weird stuff happens so we try to avoid this
    } else {
        value_str = std::to_string(value);
    }

    for (const auto& i : arr) {
        if (i == value_str) {
            return true;
        }
    }
    return false;
}

template <typename varType>
varType check_input(const std::vector<std::string>& string_arr, const
std::string& first_prompt, const std::string& try_again)
{
    std::cout << first_prompt;
    varType input;
    while (!(std::cin >> input) || !isValueInArray(input, string_arr)) {
        std::cerr << try_again;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n')
    }
    return input;
}

#endif
e_prevost@ares:~/Portfolio_1/Lab 2$ CPP driver_2 seqlist
driver_2.cpp***
seqlist.cpp...
seqlist.cpp: In function ‘void
customBubbleSort(std::vector<std::__cxx11::basic_string<char>
>&)’:
seqlist.cpp:9:21: warning: conversion
from ‘std::vector<std::__cxx11::basic_string<char>
>::size_type’ {aka ‘long unsigned
int’} to ‘int’ may change value
[-Wconversion]
    9 |     int n = vec.size();
      |             ~~~~~~~~^~
seqlist.cpp: In function ‘std::string
convertLineToString(std::string, std::string)’:
seqlist.cpp:107:50: warning:
comparison of integer expressions of different signedness:
‘std::__cxx11::basic_string<char>::size_type’ {aka
‘long unsigned int’} and ‘int’
[-Wsign-compare]
  107 |                     if(!b.empty() && b.find(',')
  == -1){
      |
      ~~~~~~~~~~~~~^~~~~
seqlist.cpp:114:50: warning: logical
not is only applied to the left hand side of comparison
[-Wlogical-not-parentheses]
  114 |             !problem_number_vector[i].find(',')
  == -1){
      |
      ^~
seqlist.cpp:114:13: note: add
parentheses around left hand side expression to silence this warning
  114 |
  !problem_number_vector[i].find(',')  == -1){
      |
      ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      |              (
      )
seqlist.cpp:114:50: warning: comparison
of constant ‘-1’ with boolean expression is always
false [-Wbool-compare]
  114 |              !problem_number_vector[i].find(',')
  == -1){
      |
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~


e_prevost@ares:~/Portfolio_1/Lab 2$ ./driver_2.out
Please Input your problem set's name: 'banana
Please Input your problem set: 1-4
Do Problems 1, 2, 3 and 4 of bananae_prevost@ares:~/Portfolio_1/Lab 2$ ./driver_2.
Please Input your problem set's name: banban
Please Input your problem set: 1,20-25
Do Problems 1, 20, 21, 22, 23, 24 and 25 of banbane_prevost@ares:~/Portfolio_1/Lab
Please Input your problem set's name: "banana"
Please Input your problem set: 1-100,50-150
Do Problems 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
e_prevost@ares:~/Portfolio_1/Lab 2$ ls
driver_2.cpp  driver_2.out  input_prot.h  seqlist.cpp  seqlist.h  seqlist.info  se
e_prevost@ares:~/Portfolio_1/Lab 2$ cat seqlist_tpq.txt
e_prevost@ares:~/Portfolio_1/Lab 2$ cat seqlist_tpq.txt
1. String



2. I can use a string searching function

or manually do it myself by looping through

the string looking for a " or '.



3. Loop through the string and create

new string that contains all letters except

letters that are " or '
```

```
4. create a numeric function to

input an item into the list based on

if it is less than the item in front of it

(goes to end if no #) and greater than the

# before it (goes to front if no #).


5. In this numeric insert function from problem 4

we can have a separate check for duplicates before insertion.

This separate check is a duplicate check after we sorted the

numbers from smallest to largest. The duplicate check just

then becomes looping through the problem list and checking if

the number already exists.


6. For our cout statement we will have a helper variable

that will keep track of how many chars we outputted.

once we reach 70 chars we std::endl then repeate_prevost@ares:~/Portfolio_1/Lab 2$
exit

Script done on 2024-09-15 13:14:52-05:00 [COMMAND_EXIT_CODE="0"]
```