```
Script started on 2024-09-15 12:58:15-05:00 [TERM="xterm-256color" TTY="/dev/pts/3'
e_prevost@ares:~/Portfolio_1/Lab 1$ pwd
/home/students/e_prevost/Portfolio_1/Lab 1
e_prevost@ares:~/Portfolio_1/Lab 1$ show-code input_prot.info
Unknown class/file type!  Please have your teacher request a group update...
e_prevost@ares:~/Portfolio_1/Lab 1$ cat input_prot.info
****************

Robert Prevost

CSC 122 W01



Input Protection Lab



Takes in input and runs through library function to recursively validate

inputs until a correct input is met.



Base Level: Level 2

Added Bonus: +1 Level (only two overloaded functions)

Total Level: Level 3

****************e_prevost@ares:~/Portfolio_1/Lab 1$ show-code input_prot.h


input_prot.h:
```

```cpp
#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)
```

```cpp
{
    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 1$ show-code driver.cpp


driver.cpp:


#include <iostream>
#include "input_prot.h"
#include <string>
using namespace std;
int main() {
    std::vector<std::string> arr = {"a", "b", "c", "d", "e"};

    string choose = "Choose a double:\n";
    string tryagain = "Try again:\n";

    string chooseL = "Choose a long:\n";

    string chooseC = "Choose a char:\n";
    check_input<double>(0, choose, tryagain, 5);
    check_input<double>(choose, tryagain,5);
    check_input<double>(0, choose, tryagain);
    check_input<long>(chooseL, tryagain);
    check_input<char>(arr, chooseC, tryagain);
    return 0;
}
e_prevost@ares:~/Portfolio_1/Lab 1$ CPP driver input_prot
driver.cpp***


e_prevost@ares:~/Portfolio_1/Lab 1$ ./driver.out
Choose a double:
10000
Try again:
30000000000
Try again:
3
Choose a double:
10000
Try again:
2
Choose a double:
-1
Try again:
10
Choose a long:
105105
Choose a char:
qwepqwoe
Try again:
jhjihjijihg
Try again:
a
```

```
e_prevost@ares:~/Portfolio_1/Lab 1$ cat input_prot_tpq.txt
TPQ

1. using a string

2. if the strings aren't changed we can pass by reference. To pass a

string through a function there needs to be a #include in all files.

3. We can pass the list through reference if they aren't going be changed

there. We can either use an array or vector. If we use a vector all files

have to have a pound include vector.

4. One function might need a min or max if we are specifying that. All the

values that aren't changed should be passed by reference.

5. the value being returned is the value that the author is querying for.

Since I used a template function the value being returned is the value

that was called with the function.

6. We need to check if it caused any errors in the cin. This can be done a

multitude of ways (try, catch for example). I used an error checking way I

learned in CSC 121 which involves looping through a cin.fail() statement.

We also need to check if it passes min and max if that is specified.

7. based on the parameters inputted.

8. ifndef define endif loop.

9. a pound include of the library. When compiling make sure to include all

file names including the library

10. A traditional library contains a header and an implementation file.

For my library I avoided using a header because it caused issues with my

VS code compiler. Since I did not have a header file I pound included the

implementation file straight up. But if I did have a header file I would

pound include the header file. e_prevost@ares:~/Portfolio_1/Lab 1$ exit
exit

Script done on 2024-09-15 12:59:55-05:00 [COMMAND_EXIT_CODE="0"]
```