```
Script started on Thu 16 Feb 2017 10:13:49 AM CST
\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ pwd
\033[K
/home/students/b_pepa/CSC122/inputValidation
\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ cat input_validation.info
Brandon Pepa
CSC122-001
Oops... Shall we try again?
Lab

(level 2)
(level 1)
        protect inputs from user stupidity

**(Level 3)**

Description:
        The function of this program is to create an overloaded function that
allows the user to pass a prompt, an error message, and bounds. The function
will prompt the user until they give a proper value to what the program was
looking for and use the error message passed to prompt the user. This function
is able to be used for doubles, longs, and characters.
\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ cat input_validation.h
#ifndef INPUT_VALIDATION_H_INC
#define INPUT_VALIDATION_H_INC

#include <vector>

double input_protect(const std::string & prompt, const std::string & error,
                     const double & lower_limit, const double & upper_limit);

double input_protect(const double & lower_limit,
                     const std::string & prompt, const std::string & error);

double input_protect(const std::string & prompt, const std::string & error,
                     const double & upper_limit);

long input_protect(const std::string & prompt ,    const std::string & error,
                   const long & lower_limit , const long & upper_limit);

long input_protect(const long & lower_limit,
                   const std::string & prompt, const std::string & error);

long input_protect(const std::string & prompt, const std::string & error,
                   const long & upper_limit);

long input_protect(const std::string & prompt, const std::string & error,
                   const std::vector<long> & value_set);

char input_protect(const std::string & prompt, const std::string & error,
                   const std::string & value_set);
#endif
\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ cat input\033[K\033[K\033[K\033[K\033[Knput_validation.cpp
#include <iostream> //for cin and cout
#include <vector> //use of vectors for list of items
using namespace std;

double input_protect(const string & prompt, const string & error,
                     const double & lower_limit, const double & upper_limit)
{
   //temp will represent the input and the return number
   double temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         fail = false;
         cin.ignore(INT_MAX, '\n');
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //if something fails, turn "fail" to true
      if(cin.fail() ||
         temp > upper_limit || temp < lower_limit)
      {
         fail = true;
      }

   } while(fail);

return temp;
}

double input_protect(const double & lower_limit,
                     const string & prompt, const string & error)
{
   //temp will represent the input and the return number
   double temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         fail = false;
         cin.ignore(INT_MAX, '\n');
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //if something fails, turn "fail" to true
      if(cin.fail() || temp < lower_limit)
```

```
      {
         fail = true;
      }

   } while(fail);

return temp;
}

double input_protect(const string & prompt, const string & error,
                     const double & upper_limit)
{
   //temp will represent the input and the return number
   double temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         cin.ignore(INT_MAX, '\n');
         fail = false;
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //if something fails, turn "fail" to true
      if(cin.fail() || temp > upper_limit)
      {
         fail = true;
      }

   } while(fail);

return temp;
}

long input_protect(const string & prompt ,    const string & error,
                   const long & lower_limit , const long & upper_limit)
{
   //temp will represent the input and the return number
   long temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         fail = false;
         cin.ignore(INT_MAX, '\n');
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //if something fails, turn "fail" to true
      if(cin.peek() == '.'  || cin.fail() ||
         temp > upper_limit || temp < lower_limit)
      {
         fail = true;
      }

   } while(fail);

return temp;
}

long input_protect(const long & lower_limit,
                   const string & prompt, const string & error)
{
   //temp will represent the input and the return number
   long temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         fail = false;
         cin.ignore(INT_MAX, '\n');
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //if something fails, turn "fail" to true
      if(cin.peek() == '.' || cin.fail() ||
         temp < lower_limit)
      {
         fail = true;
      }

   } while(fail);

return temp;
}

long input_protect(const string & prompt, const string & error,
                   const long & upper_limit)
{
```

```
   //temp will represent the input and the return number
   long temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         fail = false;
         cin.ignore(INT_MAX, '\n');
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //if something fails, turn "fail" to true
      if(cin.peek() == '.' || cin.fail() ||
         temp > upper_limit)
      {
         fail = true;
      }

   } while(fail);

return temp;
}

long input_protect(const string & prompt, const string & error,
                   const vector <long> & value_set)
{
   //temp will represent the input and the return number
   long temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //variable used in for loop when checking if it matches a value
   bool check;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         fail = false;
         cin.ignore(INT_MAX, '\n');
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //Changes check to true if any values match the vector input
      check = false;
      for(unsigned i = 0; i + 1 <= value_set.size(); i++)
      {
         if(value_set[i] == temp)
         {
            check = true;
         }
      }

      //if something fails, turn "fail" to true
      if(cin.peek() == '.' || cin.fail() || !check)
      {
         fail = true;
      }

   } while(fail);

return temp;
}

char input_protect(const string & prompt, const string & error,
                   const string & value_set)
{
   //temp will represent the input and the return character
   char temp;

   //the fail variable represents a failure in the user input
   //either it's out of range, or it's not a valid input if it
   //turns to true
   bool fail = false;

   //variable used in for loop when checking if it matches a value
   bool check;

   //loop until it doesn't fail
   do
   {
      //Whenever the input fails, do this
      if(fail)
      {
         cerr << error;
         cin.clear();
         fail = false;
         cin.ignore(INT_MAX, '\n');
      }

      //prompt the user and store their input in temp
      cout << prompt;
      cin >> temp;

      //Changes check to true if any values match the vector input
      check = false;
      for(unsigned i = 0; i + 1 <= value_set.length(); i++)
      {
         if(value_set[i] == temp)
         {
            check = true;
         }
      }

      //if something fails, turn "fail" to true
```

```
      if(cin.fail() || !check)
      {
         fail = true;
      }

   } while(fail);

return temp;
}
\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ cat inpu
t\033[K033[K033[K033[K033[K033[Kdriver.cpp
/* Brandon Pepa
 * Input validation driver
*/

#include <iostream>
#include <vector>
#include "input_validation.h"
using namespace std;

void print(long a);
void print(double a);
void print(char a);




int main(void)
{
//Initialize the vector I use for a list of numbers to choose from
//Is there an easier or nicer looking way to do this????
  vector <long> z;
  z.push_back(1);
  z.push_back(2);
  z.push_back(3);
  z.push_back(5);
  z.push_back(7);
  z.push_back(11);
  z.push_back(13);
  z.push_back(17);

//Test all of the Double Return type input protection
   print( input_protect("\nEnter a double between 10 and -10: ",
                        "\nPlease enter a valid number",
                        static_cast<double> (-10.0), static_cast<double> (10.0)) );

   print( input_protect(static_cast<double> (0.0),
                       "\nEnter a double greater than 0: ",
                       "\nPlease enter a valid number") );

   print( input_protect("\nEnter a number less than 0: ",
                        "\nPlease enter a valid number",
                        static_cast<double> (0.0)) );

//Test all of the Integer Return type input protection
   print( input_protect("\nEnter an integer between -100 and 100: ",
                        "\nPlease enter a valid nubmer",
                        static_cast<long> (-100), static_cast<long> (100)) );

   print( input_protect(static_cast<long> (50),
                        "\nEnter an integer greater than 50: ",
                        "\nPlease enter a valid nubmer") );

   print( input_protect("\nEnter an integer less than 15: ",
                        "\nPlease enter a valid number",
                        static_cast<long> (15)) );

   print( input_protect("\nEnter 1,2,3,5,7,11,13 or 17: ",
                        "\nPlease enter a valid number", z) );

//Test the character return type input protection
   print( input_protect("\nWould you like to continue? ",
                        "\nPlease enter Y/N only! \a\a\a",
                        "YyNn") );
}




void print(long a)
{
   cout << "Your value entered is " << a << endl;
}

void print(double a)
{
   cout << "Your value entered is " << a << endl;
}

void print(char a)
{
   cout << "Your value entered is " << a << endl;
}

\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ ./prog
\033[K033[K033[K033[Kdriver.out

Enter a double between 10 and -10: 11.0

Please enter a valid number
Enter a double between 10 and -10: 8
Your value entered is 8

Enter a double greater than 0: -3

Please enter a valid number
Enter a double greater than 0: asdf

Please enter a valid number
Enter a double greater than 0: 2.5
Your value entered is 2.5

Enter a number less than 0: 15

Please enter a valid number
Enter a number less than 0: -3.2
Your value entered is -3.2

Enter an integer between -100 and 100: 3.5

Please enter a valid nubmer
Enter an integer between -100 and 100: 103

Please enter a valid nubmer
Enter an integer between -100 and 100: -102

Please enter a valid nubmer
Enter an integer between -100 and 100: 54
```

```
Your value entered is 54

Enter an integer greater than 50: 35

Please enter a valid nubmer
Enter an integer greater than 50: 75
Your value entered is 75

Enter an integer less than 15: 16

Please enter a valid number
Enter an integer less than 15: 14
Your value entered is 14

Enter 1,2,3,5,7,11,13 or 17: 16

Please enter a valid number
Enter 1,2,3,5,7,11,13 or 17: 17
Your value entered is 17

Would you like to continue? f

Please enter Y/N only! \007\007\007
Would you like to continue? N
Your value entered is N
\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ cat inpu
t_validatoin\033[K033[K033[Kion.tpq
Thought Provoking Questions:

Q1.How can you pass a prompt or error message to a function as an argument

A: The messages can be passed simply as strings (either the programmer uses a
   literal string or passes a string variable)

Q2.How do you pass a string to a function? Will the strings need to be changed
  What care do you need to take for these arguments, then?

A: A literal string passed, or a string variable. The function can let the
   user choose how to format the prompt and error so the strings don't need
   to be changed. Since the strings are not changed in the function, they
   should be passed as constant call by reference.

Q3.How do you pass a list of values to a function? Will the elements need to be
   changed here? what care do you need to take for these arguments then?

A: A list of values can either be passed as a vector or an array (I used a vector
   because I am more familiar with vectors). The element won't be changed so it
   can be passed as a constant call by refernce.

Q4.What other arguments does each function take? Are the changed? What special
   care should you take with them?

A: All the arguments can be passed as constants because they're not altered.
   the function will take a prompt, an error message, and some type of limit
   (either lower, upper, both, or a list)

Q5.What value is returned by your functions? What type is it and what does it
   represent?

A: The value returned by the function is the same as the data type of the limits
   passed (double, long, char). The value represents a the input of the user
   that was valid according to the parameters of the validation (and correct
   data type)

Q6.What care does a caller of your function have to take with this return value?
   (Can they immediately assume it is a valid entry?)

A: The caller can immediately assume it's a valid entry because the input
   validation function should have taken care of any invalid values. the caller
   either has to assign it directly to a variable or can print it directly
   (in the case of my driver)

Q7.How does the compiler distinguish which of your functions is being used for
   a particular call? (They ALL have the same name, after all...)

A: The compiler can check the data type of the limits in order to distinguish the
   function. The upper limit and lower limit functions can be distinguished by
   position of the limit in the arguments for simplicity (only adds a little bit
   of complexity when calling the function... oh well) functions with an upper
   and lower limit will have an extra parameter compared to the other 3 functions
   of the same data type

Q8.How do you protect your library from being circularly included?

A: In the header file use ifndef then define to protect from circular inclusion.
   this will stop the "circle" if it sees it is defined already

Q9.What changes are needed in your main application (the test application here) to
   get it to work with the library? What about the compiling process?

A: In order to get the main application to work with the library the header file
   needs to be #included and when compiling the libary file will need to be
   compiled alongside the driver or whatever application being used.

Q10.How many files does your libary consist of? What are the? Which one(s) do you
   #include?

A: The library consists of the interface file (.h) and the implementation file (.cpp)
   only the .h file will be included
\033]0;b_pepa@mars:~/CSC122/inputValidation\007[b_pepa@mars inputValidation]$ ex\033[Kx
it
exit

Script done on Thu 16 Feb 2017 10:18:24 AM CST
```