User Input Errors

When writing programs for unknown users, there is a need to handle invalid input, so you always have the program doing exactly what you want it to. Whether it is a value you cannot use, such as a number causing division by zero, or the wrong type of input, such as a char when you were expecting an int. Your code needs to address these issues.

Handling Out-of-Bounds values

The easiest form of error checking, and the most commonly needed, is the ability to prevent your program from performing operations with values that are out of the bounds of your program. Out-of-bounds includes any values that cause invalid computations. Some examples are:

- Menu choices that are not available
- Causing a division by zero
- Invalid solutions from equations in your program, such as negative values when only positive values are logical
- Input that doesn't make sense, such as negative age values

There are several options for handling out-of-bounds input. Some examples are:

- End the program
- Skip that part of the program and continue
- Prompt the user again, nicely

Of these options, the most preferable is to prompt the user again. To do this, we can use a while loop. Here is an example of a function in C++ using a while loop to ask for a valid response:

```
int getNumber()
{
  int num;
  cout<<"Please enter a number from 1-10: ";
  cin>>num;
  while(num>10 || num<1)          //checking if number is valid (1-10)
  {
    cout<<"That is not a valid response."
        <<"Please enter a number from 1-10: ";
    cin>>num;
  }
  return num;
}
```

Handling Illegal Array Indicies

When working with arrays (vectors, strings), you need to be careful not to access a part of the array that doesn't exist, either before the start of the array, or past the end of it. This generally can be handled with a simple `if` check. In this example, the function receives a `vector` of positive `ints`, and makes sure that you can access the `array` at that index:

```
int getNumFromArray(int index, const vector<int> &myArray)
{
  int len = myArray.size();
  if(index<len && index>=0)      //checking if index is too large or small
    return myArray[index];
  return -1;
}
```

Intro Computer Programming
Error Checking

Handling Extraneous Input

Sometimes, you want the user to input multiple data, it could be names or numbers, it doesn't matter. What does matter is if they input too much data. This can cause your programs to be taking in the wrong kind of input in the wrong place, possibly leading to catastrophic failures. To avoid this problem, we need to clear the input buffer. The buffer stores all keyboard input until it is read by your program using cin. To clear the buffer, and to get rid of input on a line after the requested data, we will use the istream member function, istream& ignore(int nCount=1, int delim=EOF). "nCount" is the maximum number of characters that it will ignore and "delim" is the character that, if found, will stop the "ignoring". It can be used in this manner:

```
void getInfo(string &first, string &last, int &num1, int &num2)
{
  cout<<"Please enter a name: ";
  cin>>first>>last;
  cin.ignore(80, '\n');                  //will skip up to 80 extra characters
  cout<<"Please enter 2 numbers: ";
  cin>>num1>>num2;
  cin.ignore(80, '\n');                  //will skip up to 80 extra characters

  cout<<"Name: "<<first<<" "<<last<<"\nNumbers: "<<num1<<", "<<num2<<'\n';
}
```

Here is a sample run:

```
Please enter a name: Joe Schmoe John Doe
Please enter 2 numbers: 1 2 3 4
Name: Joe Schmoe
Numbers: 1, 2
```

Handling Illegal Input Types

You probably have written a program that receives an int as input, and then someone entered a double or a char. While it may look like that situation is hopeless, there are ways to prevent the user from crashing your program in this way.

Handling doubles

This is probably the easiest type to check for. To do this, you need to input all of your ints as doubles. To do this we will use a technique called type-casting. Type-casting forces the computer to convert one type of data to another. While type-casting between strings and numbers does not work very well, typecasting between different number types can be very useful. Here is an example:

```
int getInt()
{
  double answer;
  cin>>answer;
  return int(answer);          //answer is type-cast to an int
}
```

Intro Computer Programming
Error Checking

Handling chars and strings

You might have noticed that if you accidentally (or purposely) input a char or a string instead of an int, your program goes into an infinite loop. To fix this, we will input all data as strings, and convert it as necessary. Here is an example:

```
int getInt()
{
  string answer;
  getline(cin, answer);
  return atoi(answer.c_str());      //temp is converted to an int
}
```

You can now combine all the error checking techniques you have learned to write effective programs, not to be deterred by any erroneous data.