ACL Wildcard Mask Parser

One of the things that students who fail the **Final Skills Test (https://swinburne.instructure.com/courses/61531/assignments/631613)** often get wrong is that they incorrectly calculate the wildcard mask when writing their ACLs.

In order to aid in your learning of how to correctly write an ACL wildcard mask, please find below a program you can use to help you understand what Wildcard masks achieve

Installing Python | Wildcard Parser Program | Using the Wildcard Parser | Understanding your Mistakes

Using the Wildcard Parser

This tab includes a brief overview of the user interface of the program

User Input - Providing the ACL host range statement

When first executed, the program will display the following to the screen and wait for you to enter an option:

```
------------------------------------------------------------------------------------------------
--
Please enter a valid IP address range for a Cisco ACL. Valid options include:
  host a.b.c.d    - matches only the IP address a.b.c.d
  any             - matches all IP addresses (0.0.0.0/0 or 0.0.0.0-255.255.255.255
  a.b.c.d w.x.y.z - match IP addresses when wildcard mask w.x.y.z is applied to IP address a.b.c.
d
  q               - quit program
User input:
```

You may exit the program by typing 'q' and pressing enter. Otherwise you can enter the range specifier in any of the three allowable methods used for a Cisco ACL

Checking the Range Specifier

If you enter an invalid range specifier that does not exactly match the allowed formats, the program will detect the invalid input, print an error message, and return to the main menu

```
User input:host 300.300.300.300


ERROR: Badly formed ip range match statement (host 300.300.300.300)
```

Displaying Results

When a valid range specifier is provided, the application will output:

- Provided IP address and Wildcard mask in both dotted decimal and binary notation (note that ranges entered using **host** and **any** will be translated into equivalent **a.b.c.d w.w.w.w** statements

- A list of all matched addresses. Each item will contain one set of continuous IP addresses and be displayed as both a subnet and an IP address range (first address and last address)
- If the continuous IP address range consists of only one address (**/32**), then the last address in the range will not be printed

For example, to match all IP addresses in **192.168.0.0/24**, you would use the Wildcard mask of **0.0.0.255**. In this case the program will output:

```
User input:192.168.0.0 0.0.0.255

Processing Input: 192.168.0.0 0.0.0.255

 IP Address:    192.168.0.0    :   binary(11000000101010000000000000000000)
 Wildcard Mask: 0.0.0.255      :   binary(00000000000000000000000011111111)

 Matched IP Addresses
   192.168.0.0/24     : 192.168.0.0     - 192.168.0.255
```

Here we can see that the wildcard mask contains 8 1-bits for the last octet, that all addresses in **192.168.0.0/24** are matched, and that the matched address range is **192.168.0.0-192.168.0.255**

Similarly, if we wish to get all the odd IP addresses in the subnet **192.168.1.0/28**, we would use an IP address of **192.168.1.1** and a Wildcard mask of **0.0.0.14**. In this case, the output is:

```
User input:192.168.1.1 0.0.0.14

Processing Input: 192.168.1.1 0.0.0.14

 IP Address:    192.168.1.1    :   binary(11000000101010000000000100000001)
 Wildcard Mask: 0.0.0.14       :   binary(00000000000000000000000000001110)

WARNING: Possible mistake entering rule, more than one continuous range of addresses. If you are
not trying to do something like odd or even addresses, this
is likely wrong

 Matched IP Addresses
   192.168.1.1/32     : 192.168.1.1
   192.168.1.3/32     : 192.168.1.3
   192.168.1.5/32     : 192.168.1.5
   192.168.1.7/32     : 192.168.1.7
   192.168.1.9/32     : 192.168.1.9
   192.168.1.11/32    : 192.168.1.11
   192.168.1.13/32    : 192.168.1.13
   192.168.1.15/32    : 192.168.1.15
```

Note that the output now contains 8 lines, one for each of the 8 odd IP addresses in the subnet

You should also note the **WARNING** message. It is unlikely that you will be required to create a wildcard mask that does not encompass an entire subnet range, so unless this is what you are trying to achieve, this is an indication that you have likely made an error in calculating the wildcard mask

An example of this is below, again when trying to match 192.168.0.0/24. A common mistake made by students in this case is to use a wildcard mask of 0.0.1.0

```
User input:192.168.0.0 0.0.1.0

Processing Input: 192.168.0.0 0.0.1.0

 IP Address:    192.168.0.0    :   binary(11000000101010000000000000000000)
 Wildcard Mask: 0.0.1.0        :   binary(00000000000000000000000100000000)

WARNING: Possible mistake entering rule, more than one continuous range of addresses. If you are
not trying to do something like odd or even addresses, this
```

```
is likely wrong

 Matched IP Addresses
   192.168.0.0/32     : 192.168.0.0
   192.168.1.0/32     : 192.168.1.0
```

Note that the first sign of error is that the binary wildcard mask does not contain a string of 0's followed by a string of 1's. Also note that it only matches one IP address in your subnet, along with an IP address in the next subnet