<Report on Regular Expression practice>

2017320233 김채령

Environment

I used C language as a programming language with PCRE2 library and Visual Studio 2017 Community version as IDE.

Algorithm

I implemented my program mainly using a for-loop and regex pattern with PCRE2 library which determines whether to accept the input string or not

We've got 3 problems which are phone number classifier, log classifier, and specific image file classifier. Thus, by implementing the for-loop with int p, we repeat the for-loop block for 3 times, each for the problem. In the loop, p could be either 0, 1, or 2 and this implies which problem the program will handle at that time. Therefore, I inserted switch block to select each proper regex pattern for each problem. For each problem, we take 'num' number of inputs and for num times we determine whether each input string matches the regex or not. Here, in each problem for taking 'num' number of inputs and checking them, while-loop helps the processing. Additionally, for each input string, if it matches to the regex we print "Y", otherwise "N".

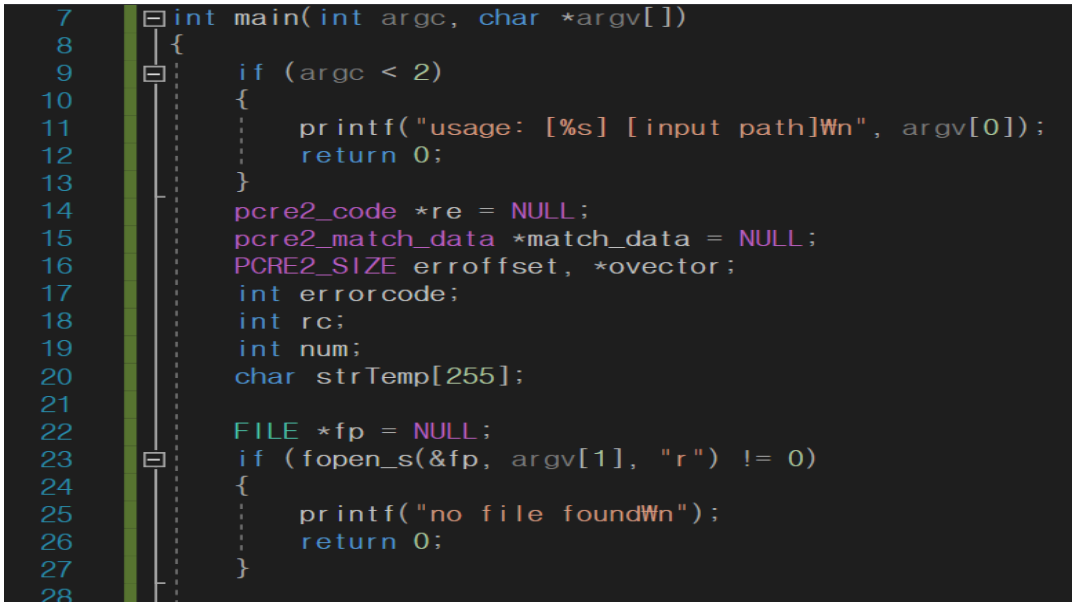
Explanation of Code

```
1   #define PCRE2_CODE_UNIT_WIDTH 8
2   #include "pcre2.h"
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
```

1: define PCRE2_CODE_UNIT WIDTH to 8.

2: include "pcre2.h" header file to use pcre2_(..) codes.

3-5: include header files.



```
7   int main(int argc, char *argv[])
8   {
9       if (argc < 2)
10      {
11          printf("usage: [%s] [input path]₩n", argv[0]);
12          return 0;
13      }
14      pcre2_code *re = NULL;
15      pcre2_match_data *match_data = NULL;
16      PCRE2_SIZE erroffset, *ovector;
17      int errorcode;
18      int rc;
19      int num;
20      char strTemp[255];
21
22      FILE *fp = NULL;
23      if (fopen_s(&fp, argv[1], "r") != 0)
24      {
25          printf("no file found₩n");
26          return 0;
27      }
28
```

7: main function - parameters in the parentheses are to send arguments of a program. To count of the arguments(programs) we use 'argc', and 'argv' is the argument vector.

9-13: This if statement handles the input error in the case of argc being smaller than 2 since when nothing passed to the argument of the program, argc becomes 1.

14: *re - a pointer to the compiled pattern initialized as NULL.

15: *match_data - a data block for storing the match result.

16: erroffset – for offset. (indicates where the error rose)

(ovector - a vector of offsets, defines the matched part.)

17: errorcode – get error message.

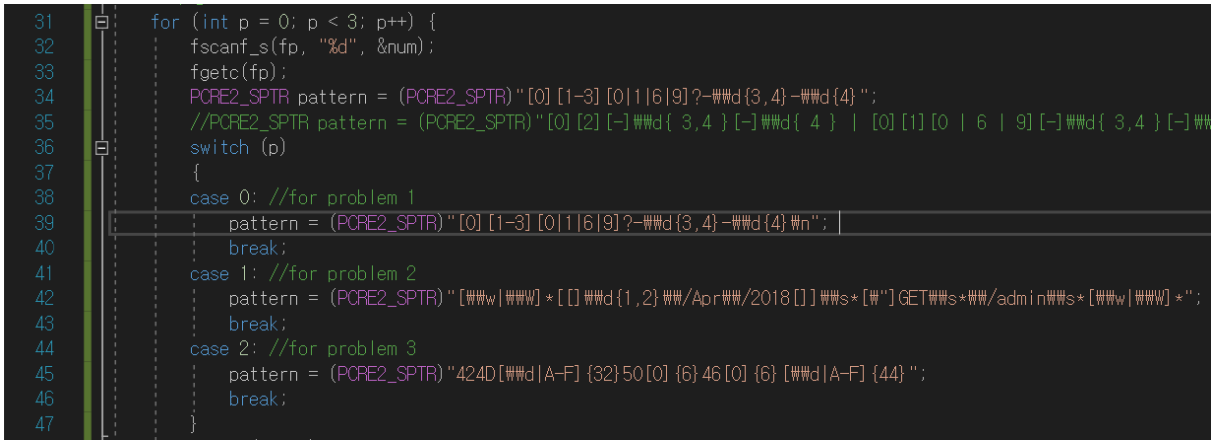
18: rc – for return value of pcre2_match. If no match, zero or negative.

19: num - count the number of cases for each problem.

20: strTemp - the string read from the input file.

22: *fp – declare the file pointer.

23-27: handling error - If the file does not exist, show error.



```
31  for (int p = 0; p < 3; p++) {
32      fscanf_s(fp, "%d", &num);
33      fgetc(fp);
34      PCRE2_SPTR pattern = (PCRE2_SPTR)"[0][1-3][0|1|6|9]?-₩₩d{3,4}-₩₩d{4}";
35      //PCRE2_SPTR pattern = (PCRE2_SPTR)"[0][2][-]₩₩d{ 3,4 }[-]₩₩d{ 4 } | [0][1][0 | 6 | 9][-]₩₩d{ 3,4 }[-]₩₩
36      switch (p)
37      {
38      case 0: //for problem 1
39          pattern = (PCRE2_SPTR)"[0][1-3][0|1|6|9]?-₩₩d{3,4}-₩₩d{4}₩n";
40          break;
41      case 1: //for problem 2
42          pattern = (PCRE2_SPTR)"[₩₩w|₩₩W]*[]₩₩d{1,2}₩₩/Apr₩₩/2018[]]₩₩s*[₩"]GET₩₩s*₩₩/admin₩₩s*[₩₩w|₩₩W]*";
43          break;
44      case 2: //for problem 3
45          pattern = (PCRE2_SPTR)"424D[₩₩d|A-F]{32}50[0]{6}46[0]{6}[₩₩d|A-F]{44}";
46          break;
47      }
```

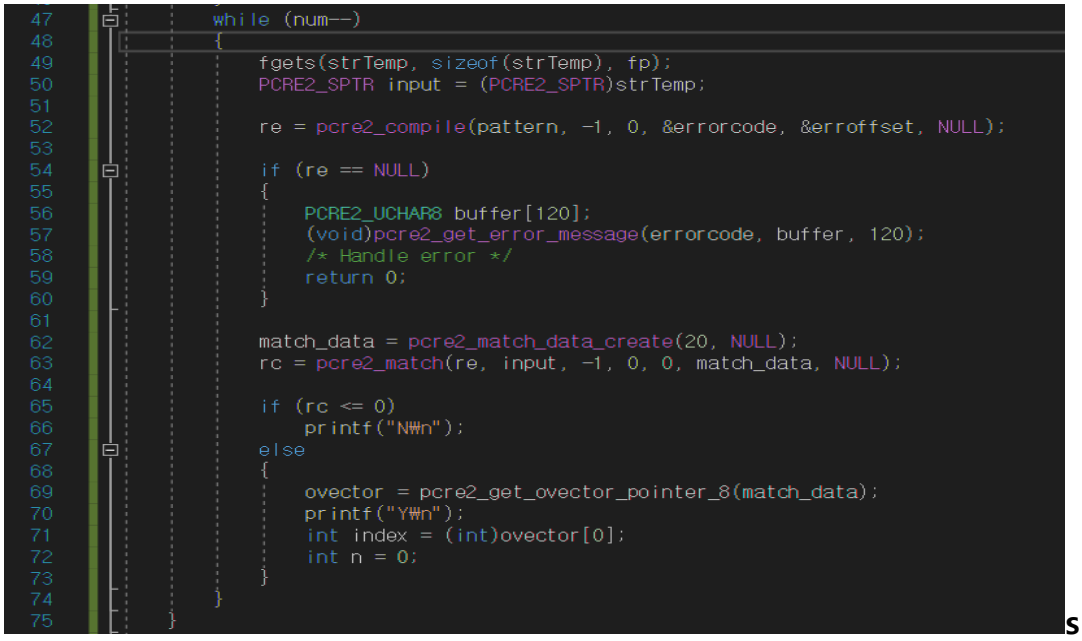
31: for-loop – repeat 3 times, each loop for each problem.

(we have 3 problems – p is for indicating each problem)

32-33: read the first line for the number of test cases, and it is assigned to num.

34: declare pattern in type PCRE2_SPTR and initialize to avoid error.

36-47: each pattern for problem 1, 2, 3 respectively assigned to pattern.



```
47      while (num--)
48      {
49          fgets(strTemp, sizeof(strTemp), fp);
50          PCRE2_SPTR input = (PCRE2_SPTR)strTemp;
51
52          re = pcre2_compile(pattern, -1, 0, &errorcode, &erroffset, NULL);
53
54          if (re == NULL)
55          {
56              PCRE2_UCHAR8 buffer[120];
57              (void)pcre2_get_error_message(errorcode, buffer, 120);
58              /* Handle error */
59              return 0;
60          }
61
62          match_data = pcre2_match_data_create(20, NULL);
63          rc = pcre2_match(re, input, -1, 0, 0, match_data, NULL);
64
65          if (rc <= 0)
66              printf("N₩n");
67          else
68          {
69              ovector = pcre2_get_ovector_pointer_8(match_data);
70              printf("Y₩n");
71              int index = (int)ovector[0];
72              int n = 0;
73          }
74      }
75  }
```

s

47: while loop – repeated for num times.

49: read the next line of the input file, assign to strTemp.

50: declare input, type casting, assign to input.

52: assign a pointer to a private data structure containing the compiled pattern, or NULL if there's an error to re.

54-60: error handling.

62: match_data – data block for holding the result of match.
63: right pattern returns zero if the vector of offsets is too small, or negative error code for no match and other errors.
65-66: print N if no match.
67-73: print Y if match.

```
77      fclose(fp);
78
79      pcre2_match_data_free(match_data);
80      pcre2_code_free(re);
81
82      return 0;
83
84  }
```

77: close the file.
79-80: free match_data and re before fininishing the program.
82: end the program.

Explanation of Regex

1) p1. Phone Number Classifier
: identifying whether the given string is a phone number or not

>> string format
A – B1 – B2
Here,
A: Area Code, one of 02, 031, 010, 016, 019
B1: Middle letters, 3 or 4 digits
B2: Last letters, 4 digits(in general)

>>Therefore, the regex for a phone number can be found like this:
(*note that ₩₩ refers to a backslash in source code)
[0][1-3][0|1|6|9]?-₩₩d{3,4}-₩₩d{4}₩n

To explain this regex more specifically, it follows like this.

For the Area Code, all of 5 possibilities start with 0 followed by 1 or 2 or 3. The area code can be done by here or it could have another letter 0 or 1 or 6 or 9. Therefore, the regex for the Area code came out to be [0][1-3][0|1|6|9]? where [1-3] refers to one appearance of 1 or 2 or 3 and the quantifier '?' means zero or one occurrence of 0 or 1 or 6 or 9.

Secondly, for the middle letters, there should be 3 or 4 occurrences of a digit. Similarly, for the last letters, there should be 4 occurrences of a digit. Thus, ₩₩d{3,4} and ₩₩d{4} become each regex for the middle letters and the last letters respectively. {3,4} here refers to between 3 or 4 occurrences(inclusive) of the digit and {4}₩n here means 4 occurrences of the digit exactly.

Finally, area code, middle letters, and the last letters are connected with dash '-' so the complete regex for a phone number turns out to be

[0][1-3][0|1|6|9]?-₩₩d{3,4}-₩₩d{4}₩n

Cf. A lot more detailed explanation for the regex:

[0]

0 matches the character 0 literally

[1-3]

A single character in the range between 1 and 3 (inclusive)

[0|1|6|9]?

? quantifier matches between zero or one occurrence

0|1|6|9 matches a single character in the list

₩₩d{3,4}

₩₩d matches a digit (equal to [0-9])

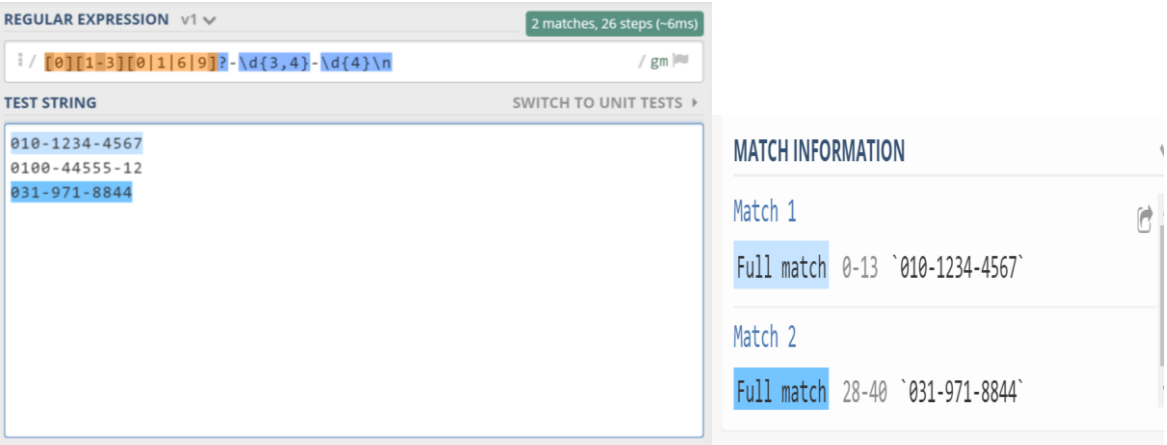
{3,4} quantifier matches between 3 and 4 times occurrences (inclusively)

₩₩d{4}₩n

₩₩d matches a digit (equal to [0-9])

{4}₩n matches exactly 4 times occurrences

>> Checking regex online https://regex101.com/ with sample inputs



2) p2. Log Classifier

: determining if the given string is the requested log in /admin on GET in April 2018

>> string format

~[date/Apr/2018] "GET /admin -

Here,

~: some combination of word and non-word characters

[date/Apr/2018] "GET /admin: date can be 1 digit or 2 digits

-: some combination of word and non-word characters

>>Therefore, the regex for the log can be found like this:

(*note that ₩₩ refers to a backslash in source code)

[₩₩w|₩₩W]*[[]₩₩d{1,2}₩₩/Apr₩₩/2018[]]₩₩s*[₩"]GET₩₩s*₩₩/adimin₩₩s*[₩₩w|₩₩W]*

To explain this regex more specifically, it follows like this.

[₩₩w|₩₩W]*, the starting and the last end of the regex, refers to zero or more occurrences of [₩₩w|₩₩W] which is any word character or any non-word character.

The middle part, [date/Apr/2018] "GET /admin , can be converted into [[]₩₩d{1,2}₩₩/Apr₩₩/2018[]]₩₩s*[₩"]GET₩₩s*₩₩/admin₩₩s* and let's consider it one by one. [[] is to match the character '['. ₩₩d{1,2} is for the date, which is 1 or 2 digits. ₩₩/ matches '/' , a slash. Apr matches literally 'Apr' which implies

April in this problem. Again, ₩₩/ refers to a slash and 2018 matches to literal '2018'. [] is to refer an appearance of ']' , and ₩₩s* means zero or more occurrences of whitespace character. To represent a single double quote, " , we use the regex [₩"] and plus, GET literally matches to 'GET'. Again, ₩₩s* implies zero or more whitespace character and ₩₩/ implies a slash. Similarly, admin is for the literal 'admin' and ₩₩s* was mentioned before.

The last part of the regex, [₩₩w|₩₩W]* is the same as the starting portion of the regex which matches to zero or more combinations of word character or non-word character.

So if we put all of those portions of regex together, the complete regex for log classifier looks like this.

[₩₩w|₩₩W]*[[]₩₩d{1,2}₩₩/Apr₩₩/2018[]]₩₩s*[₩"]GET₩₩s*₩₩/adimin₩₩s*[₩₩w|₩₩W]*

Cf. A lot more detailed explanation for the regex:

₩₩w matches any word character (equal to [a-zA-Z0-9_])

₩₩W matches any non-word character

[[]

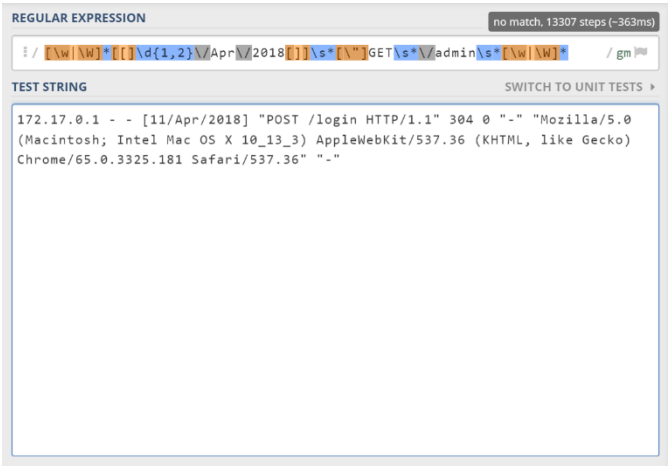
[matches the character [literally

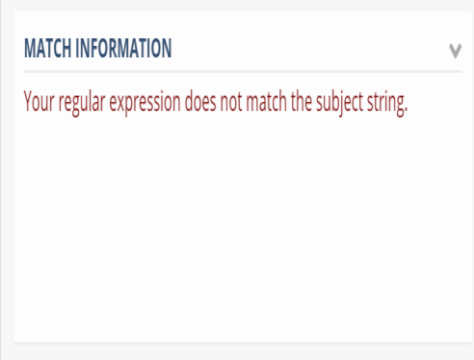
₩₩/ matches the character / literally

[₩"]

₩" matches the character " literally

>> Checking regex online https://regex101.com/ with sample inputs





3) p3. Specific Image File Classifier

: determining that the given string is the hexadecimal header string of a BMP file whose dimensions are 80*70 pixels in width and height

>> string format

424D~5000000046000000-

Here,

424D: tells us that the image file is BMP file

S: 32 digits for offset(02) – offset(17)

50000000: tells us that the width is 80 in hexadecimal

46000000: tells us that the height is 70 in hexadecimal

- : 44 digits of the total 96 characters from the end

>>Therefore, the regex for the image file can be found like this:

(*note that ₩₩ refers to a backslash in source code)

424D[₩₩d|A-F]{32}50[0]{6}46[0]{6}[₩₩d|A-F]{44}

To explain this regex more specifically, it follows like this.

Every BMP file has 0x42 and 0x4D for offset(00) and offset(01), which implies BM so, the first part of the regex becomes 424D.

From offset(02) to offset(17), there're 32 digits of hexadecimal letters which are 0-9 and A-F. In those 32 digits, several other information is contained.

The width size of the image file is given as 80, so if we invert 80 into the hexadecimal it becomes 0x50. So it's written for the offset(18) and 00 is given for the offset(19)-offset(21).

For the height size of the image file, 70 and 0x46 in hexadecimal way, 46 goes for the offset(22) and 00 for the offset(23)-offset(25).

The other remaining part with 44 characters are composed of [₩₩d|A-F] and they are for the last part out of total 96 characters.

Therefore, if we put all of those portions of regex together, the complete regex for the BMP image file classifier looks like this.

424D[₩₩d|A-F]{32}50[0]{6}46[0]{6}[₩₩d|A-F]{44}

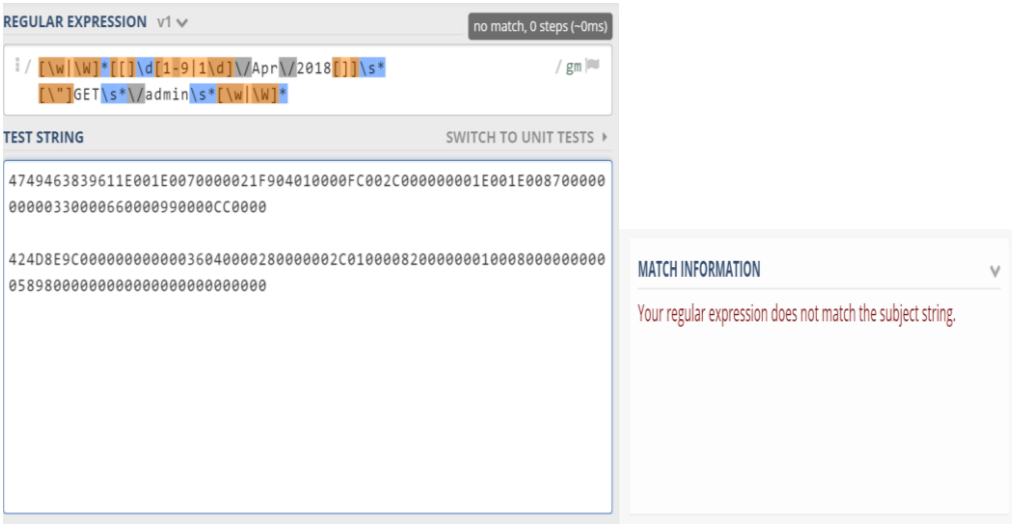
Cf. A lot more detailed explanation for the regex:

[₩₩d|A-F]

₩₩d matches a digit (equal to [0-9])

A-F matches a single character in the range between A and F (inclusive)

>> Checking regex online https://regex101.com/ with sample inputs

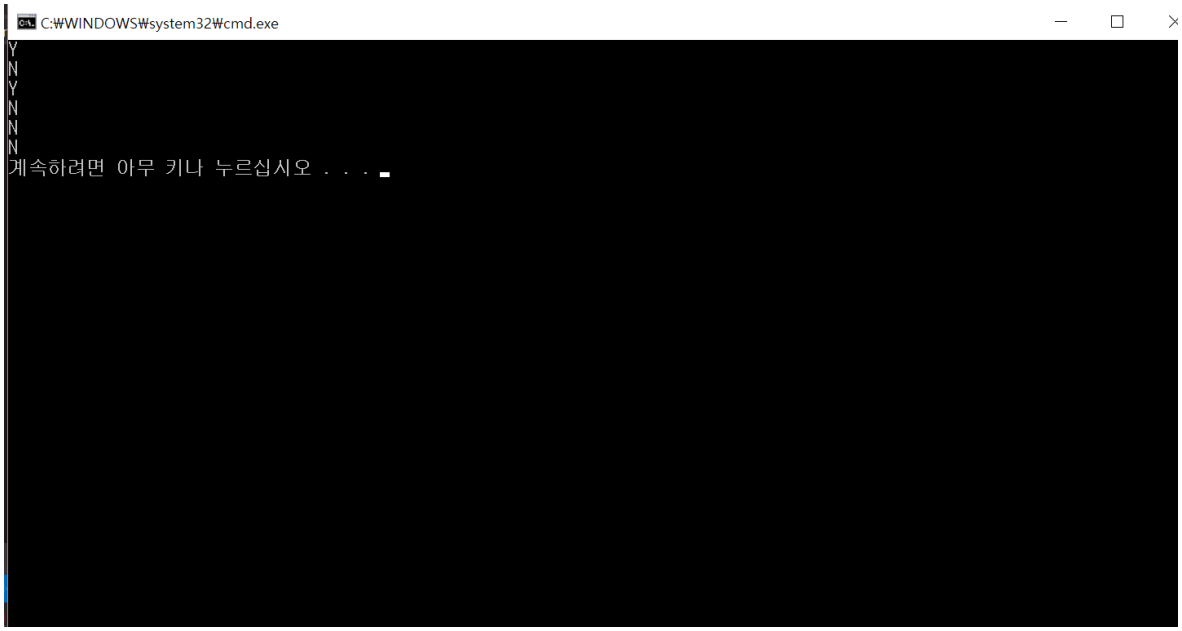


Explanation of Result

Here is the given input.txt

```
3
010-1234-4567
0100-44555-12
031-971-8844
1
172.17.0.1 - - [11/Apr/2018] "POST /login HTTP/1.1" 304 0 "-" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_13_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36" "-"
2
474946383961 1E001E0070000021F904010000FC002C000000001E001E008700000000003300006600009900000CC0000
424D8E9C0000000000003604000028000000 2C0100008200000001000800000000 0589800000000000000000000000
```

If I process the problem, the results are given like this.



For the problem 1, we got 3 inputs and the second string does not match the regex pattern in terms of area code. Therefore, it prints "N".

For the problem 2, the only input does not match the log in /admin on GET so it prints "N".

For the last problem, we took 2 inputs. The first input string violated even the offset(00) which should have been 42. Also, the second input string does not match the pattern since it's size differs.

Eventually, the result of the program is printed as Y N Y N N N.

-End😊