**Prgramming Report**

Moon Gi Jun

**Introduction to the Assignment:**

The main aim of this assignment is to help students to understand how regular expression can be practically used in computer programming. The basic structure of the code is given, and the requirement for students is to implement the regular expression for the variety of stings that have sudden restrictions. This is to classify sudden strings that are in specific form that is represented on the assignment. To implement this in code we should utilize PCRE library in code development system.

**What is PCRE library?**

PCRE is the library that help us to utilize regular expression in C language code development system. To be specific, The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5. In this assignment we are using the newest version, PCRE2, released in 2015.

**Implementation of the code:**

I implement the code with C++ with visual studio 2017 environment. PCRE sample code is given by the name of “pcre\_sample.c”. It is code that can classify the correct inputs of age of a person. It is given to show students the basic structure of utilization of PCRE. The structure of the code is simple to manipulate. Which, we can totally change the progress of the code only by changing the regular expression inside the blanket(“”).

PCRE2\_SPTR pattern = (PCRE2\_SPTR)"**Regular Expression**";

To develop the code that satisfy the requirement of our assignment, we must change sudden part of the code. Original code was only design to classify one kind of stings. However, in our assignment, we have to make the program to deal with three different type of strings. Therefore, I chose to “copy and paste” line 33-60 of the code three times, and I changed the “pattern” variable into three different types to deal with those three copied code. The names of the variable that specifically deal with each type of string classifier are pattern1(deal with phone number classifier), pattern2(deal with log classifier), pattern3(deal with specific image file classifier).

**Implementation of Regular Expression:**

During implementation “<https://regex101.com/>” was very helpful to design the expression that I want to build. It was very helpful while I test my regular expressions. However, when we are building the code in “Visual Studio” environment, we should modify the expression to let the computer recognize this expression. First, the special symbol that has specific meaning in coding such as “ ; [ ] ( ) have to add \ sign to make it as a character. Moreover, all the \ in regular expression have to be changed to \\ to let complier understand it with PCRE library. Now, let’s discuss about how I built my expressions.

1. **Phone Number Classifier**
   1. **First approach**

When we first see the basic concept of the phone number we can actually think of 3digit-4digits-4digits structure. Therefore, we can come up with the structure of this:

**\d{3}-\d{4}-\d{4}**

However, this doesn’t include any specification of area code and the number of middle letter, so I did second approach.

* 1. **Second approach**

To add the specification of area code and the number of middle letter, I change the code into this:

**([0][2][-]\d{3,4}[-]\d{4})|([0][3][1][-]\d{3,4}[-]\d{4})|([0][1][0][-]\d{3,4}[-]\d{4})|([0][1][6][-]\d{3,4}[-]\d{4})|([0][1][9][-]\d{3,4}[-]{4})**

This code include area code of 02, 031, 010, 016, 019 respectively, and it’s connected with “or”(|) operation to deal with each different case. Moreover, “\d{3,4}” was used to represent the condition of middle letters to be 3 or 4 letters. However, this still include problem because it actually accept strings of 002-234-2341 or 02-234-23411. This means the start and end of the string is not shown in the expression. Therefore we go to the third approach.

* 1. **Third approach**

To show the start and end of the string exactly, I added \G(start of match) and \Z(end of string) to each composition of number. The final code is this:

**(\G[0][2][-]\d{3,4}[-]\d{4})\Z|(\G[0][3][1][-]\d{3,4}[-]\d{4}\Z)|(\G[0][1][0][-]\d{3,4}[-]\d{4}\Z)|(\G[0][1][6][-]\d{3,4}[-]\d{4}\Z)|(\G[0][1][9][-]\d{3,4}[-]\d{4}\Z)**

This expression exactly satisfies all the possible outcome that the assignment had shown us.

1. **Log Classifier**

For log classifierI hypothesized that the string start from HTTP to the end is fixed to every log string.

* 1. **First approach**

For log classifierWhen we first see the basic concept of the log string, we can come up with the structure of this:

**\d{1,2,3}.\d{1,2,3}.\d{1,2,3}.\d{3}**

**\s-\s-\s\[\d{2}\/Apr\/2018\]\s(\")GET\s\/admin\s**

**HTTP\/1.1(\")\s304\s0\s(\")-(\")\s(\")Mozilla\/5.0(\s+)**

**\(Macintosh;\sIntel\sMac\sOS\sX\s10\_13\_3\)**

**\sAppleWebKit\/537.36\s\(KHTML.\slike(\s+)**

**Gecko\)\sChrome\/65.0.3325.181\sSafari\/537.36(\")\s(\")-(\")**

As I mentioned above the string start from HTTP is fixed. Therefore, it is very obvious to represent it as this. I used the blue colored “\s+” because it is represented in the powerpoint of the assignment that the amount of space between the lines were not exactly given.

Now let’s talk about the error that this expression has. As most people know each digit of IP address cannot go over 255, so it always have to be any digits from 0-255. Moreover, the date in April is only from 1-30. Therefore, I went to the second approach.

* 1. **Second approach**

This is the code that corrected all the error of the first approach. The red part is the corrected part.

**(\d{1,2}|[1]\d{2}|[2][0]\d|[2][1]\d|[2][2]\d|[2][3]\d|[2][4]\d|[2][5][0]|[2][5][1]|[2][5][2]|[2][5][3]|[2][5][4]|[2][5][5])([.](\d{1,2}|[1]\d{2}|[2][0]\d|[2][1]\d|[2][2]\d|[2][3]\d|[2][4]\d|[2][5][0]|[2][5][1]|[2][5][2]|[2][5][3]|[2][5][4]|[2][5][5])){3}\s-\s-\s\[([1-9]|[1]\d{1}|[2]\d{1}|30)\/Apr\/2018\]\s(\")GET\s\/admin\sHTTP\/1.1(\")\s304\s0\s(\")-(\")\s(\")Mozilla\/5.0(\s+)\(Macintosh;\sIntel\sMac\sOS\sX\s10\_13\_3\)\sAppleWebKit\/537.36\s\(KHTML.\slike(\s+)Gecko\)\sChrome\/65.0.3325.181\sSafari\/537.36(\")\s(\")-(\")**

For the IP address, I classified the number between 0-255 into 3 types. One that has only one to two digits like 0to 99(**\d{1,2}**). Another type is all the number from 100to 199(**[1]\d{2}**). The last is all the number from 200to 255(**[2][0]\d|[2][1]\d|[2][2]\d|[2][3]\d|[2][4]\d|[2][5][0]|[2][5][1]|[2][5][2]|[2][5][3]|[2][5][4]|[2][5][5]**). For the number form 200 to 255, I divided it into 20\*, 21\*, 22\*, 23\*, 24\*, 250, 251, 252, 253, 254, 255 this eleven types. (\* represent the one digit number from 0-9)

For the date, I classified the number between 1-30 into 4 types. One that has only one to two digits like 0to 9(**\d{1}**). Another type is all the number from 10to 19(**[1]\d{1}**). The third is all the number from 20to 29(**[2]\d{1}**). The Last is the number 30 itself.

However, this still include problem because it actually accept strings that include this format, but not exactly the same. In another word, adding some string in front and back would result the same. This means the start and end of the string is not shown in the expression. Therefore we go to the third approach.

* 1. **Third approach**

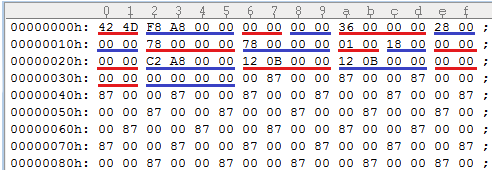
To show the start and end of the string exactly, I added \G(start of match) and \Z(end of string) to each composition of number. The final code is this:

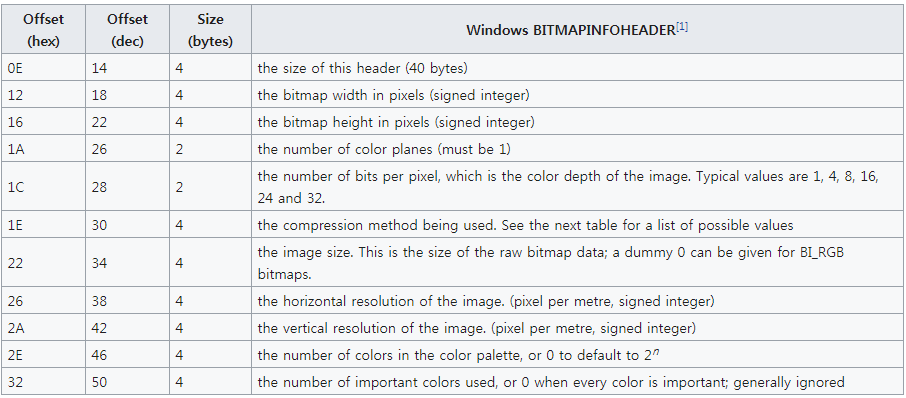
**\G(\d{1,2}|[1]\d{2}|[2][0]\d|[2][1]\d|[2][2]\d|[2][3]\d|[2][4]\d|[2][5][0]|[2][5][1]|[2][5][2]|[2][5][3]|[2][5][4]|[2][5][5])([.](\d{1,2}|[1]\d{2}|[2][0]\d|[2][1]\d|[2][2]\d|[2][3]\d|[2][4]\d|[2][5][0]|[2][5][1]|[2][5][2]|[2][5][3]|[2][5][4]|[2][5][5])){3}\s-\s-\s\[(\d{1}|[1]\d{1}|[2]\d{1}|30)\/Apr\/2018\]\s(\")GET\s\/admin\sHTTP\/1.1(\")\s304\s0\s(\")-(\")\s(\")Mozilla\/5.0(\s+)\(Macintosh;\sIntel\sMac\sOS\sX\s10\_13\_3\)\sAppleWebKit\/537.36\s\(KHTML.\slike(\s+)Gecko\)\sChrome\/65.0.3325.181\sSafari\/537.36(\")\s(\")-(\")\Z**

This expression exactly satisfies all the possible outcome that the assignment had shown us.

1. **Specific Image File Classifier**

In this part, we need to manipulate regular expressions that determine whether the given string is the hexadecimal header string of a BMP file with dimensions of 80\*70 pixels. Moreover, we are only checking, the front part of the string that is 48 beats which is comprised of 96 characters. Below is the BMP header structure. Before getting into the problem, we need to carefully consider the structure of BMP file.





According to the image and the table we can see the structure of BMP file that the string must start from 424D. Moreover, to determine the size we have to deal with two section. One is 4 bites which starts from 18 refers image width in pixel. Second is 4 bites which starts from 22 refers image lengith in pixels.

* 1. **First approach**

When we first see the basic concept of the image file we can actually think of 96 length hexadecimal string structure. Therefore, we can come up with the structure of this:

**([0-9]|[A-F]){96}**

However, this doesn’t include any specification of BMP header and size of the BMP file, so I did second approach.

* 1. **Second approach**

To add specification of BMP header and size of the BMP file, I change the code into this:

**424D([0-9]|[A-F]){32}50([0-9]|[A-F]){6}46([0-9]|[A-F]){50}**

The first part 424D represent the header string of BMP file. Moreover, after 32 6 hexadecimal character which is the totally 36, telling 2\*18, the 18th position of the file string the width of the image file follows, so I added 50 which is the transition of 80 in decimal. Moreover, after 6 hexadecimal character which is totally 44, telling 2\*22, the 22nd position of the file string the length of the image file follows, so I added 46 which is the transition of 70 in decimal. After it follows by 50 hexadecimal characters.

However, this still include problem because it actually accept strings that include this format, but not exactly the same. In another word, adding some string in front and back would result the same. This means the start and end of the string is not shown in the expression. Therefore we go to the third approach.

* 1. **Third approach**

To show the start and end of the string exactly, I added \G(start of match) and \Z(end of string) to each composition of number. The final code is this:

**\G424D([0-9]|[A-F]){32}50([0-9]|[A-F]){6}46([0-9]|[A-F]){50}\Z**

This expression exactly satisfies all the possible outcome that the assignment had shown us.

**Conclusion:**

The result of the code is exactly the same with the powerpoint. This assignment was very interesting experience to learn about how to implement regular expression in actual code. I hope there will be more projects like this.

**Thank you for reading.**