Homework#4 Report

2019320139 Choi Yun Ji

1. Environment

OS : Window 10

IDE : Visual Studio 2017

Programming language : C++

RE Library : Regex

2. Pro1

Brief Algorithm

```
make re object representing morse code
while ( there are partial strings to matched to re )
{
    convert smatch to string
    find responding character using function 'morsematch' and print it
    find next match
}
```

In regex of c++, error occurs when a regular expression becomes longer or more complex. Hence, it was almost impossible to represent a regular expression for all characters in one regular expression. Instead, possible partial strings of Morse code are extracted and responded to each character. And in c++, the blackslash should be used twice.

For extracting possible partial string, re "[-₩₩.]{1, 7}" is used.

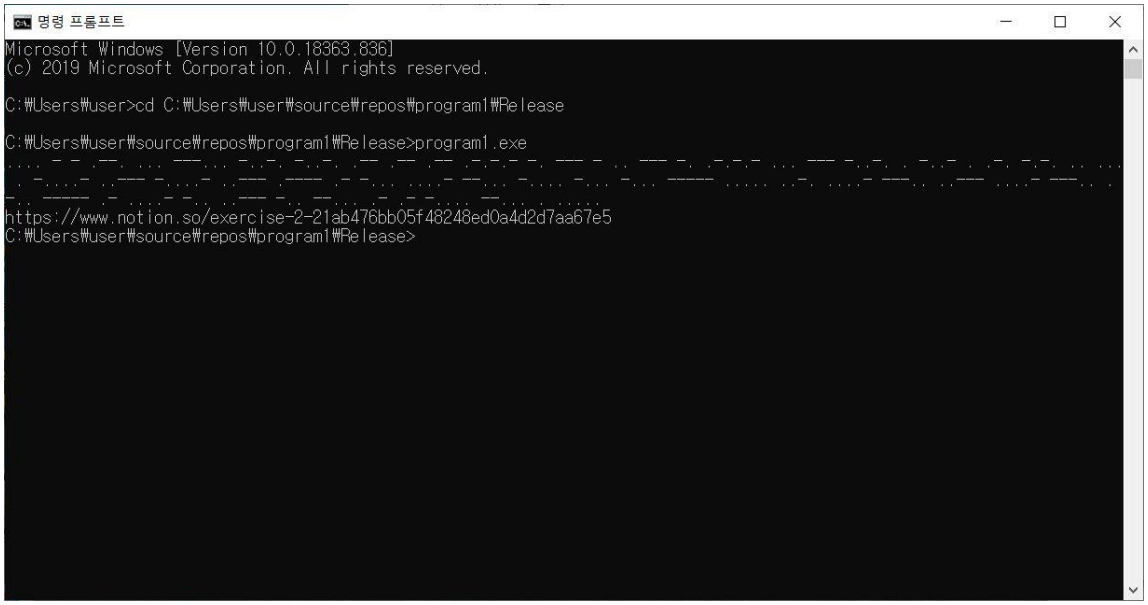
And using function 'morsematch', partial strings are responded to each character.

An explanation of code is provided as annotations of following source code.

```cs
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <iostream>
#include <regex>
#include <string>

using namespace std;

void morsematch(string str);
int main(int argc, char * argv[])
```

```
{
    string str;
    getline(cin, str); //input morse code string
    regex re("[-\\.]{1,7}");

    sregex_iterator it(str.begin(), str.end(), re); //iterator to repeating matching string to re
    sregex_iterator end;

    while (it != end) //while there are more string matched to re
    {
        smatch match = *it; //smatch object for representing a matched string
        string match_str = match.str(); //convert into string

        morsematch(match_str); //match morse code to one character
        it++; //find next match
    }
}
void morsematch(string str) //function for matching morsecode to one character
{

    if (regex_match(str, regex("\\.-"))) //if str matches to regex("\\.-"), print 'a'
        printf("a");
         .
         .
         .
         . skip
         .
         .
    else if (regex_match(str, regex("\\.-{2}\\.-\\."))) //if str matches to regex("\\.-{2}\\.-
\\."), print '@'
        printf("@");
}
```

Resulting output



3. Pro2

Brief algorithm

```
make re representing social security number and email
open input text file
while ( get a string line from input text file until there is no more string )
{
    while ( there is a partial string matched to security number pattern )
    {
        Increase number of securitynumber
        find next match
    }
    while ( there is a partial string matched to email pattern)
    {
        Increase number of email
        find next match
    }
}
Print the number of social security number of email
```

For finding security number, regular expression "₩₩s((98|99|00|01)((0(1|3|5|7|8)|1(0|2))([0-2]₩₩d|30|31)|(0(4|6|9)|11)([0-2]₩₩d|30)|02([0-1]₩₩d|2[0-8]))|000229)-([1-4][₩₩d]{6}₩₩s " is used.

And for finding email, regular expression

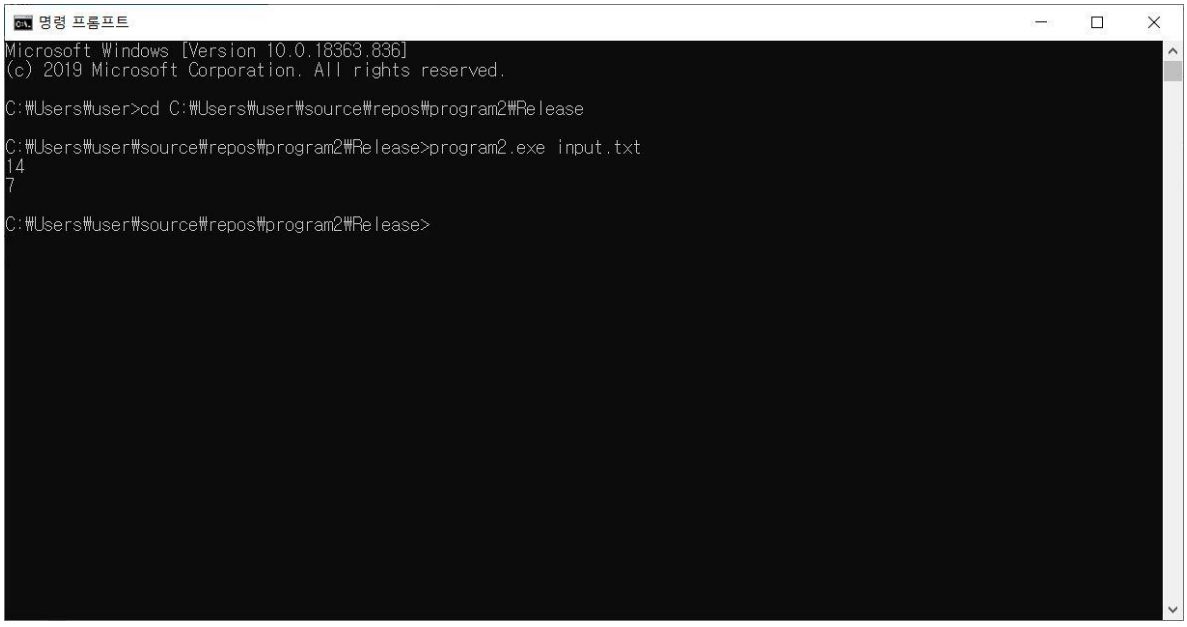
"₩₩s([A-Za-z0-9]+)@([a-zA-Z]+)₩₩.ac₩₩.kr₩₩s" is used.

An explanation of code is provided as annotations of following source code.

```
#include <stdio.h>
#include <regex>
#include <string>

using namespace std;

int main(int argc, char * argv[])
{
    int numofsecuritynum = 0, numofemail = 0; //variable for counting personal information

    regex securitynum("\\s((98|99|00|01)((0(1|3|5|7|8)|1(0|2))([0-2]\\d|30|31)|(0(4|6|9)|11)([0-
2]\\d|30)|02([0-1]\\d|2[0-8]))|000229)-([1-4])[\\d]{6}\\s");
    regex email("\\s([A-Za-z0-9]+)@([a-zA-Z]+)\\.ac\\.kr\\s");

    FILE * fp1 = fopen(argv[1], "r"); //open the input file in the path specified by argument
    char s[1001]; //char array for input string
```

```
    while (fgets(s, 1001, fp1) != NULL) //while there are strings to call in input file
    {
        string str = string(s);
        sregex_iterator iter1(str.begin(), str.end(), securitynum); //iterator to repeat matching st
ring to securitynumber pattern
        sregex_iterator end;

        while (iter1 != end) //while there is a partial string matched to securitynumber pattern
        {
            numofsecuritynum++; //increase numberofsecuritynumber
            iter1++; //find next match
        }

        sregex_iterator iter2(str.begin(), str.end(), email); //iterator to repeat matchig string to
 email pattern

        while (iter2 != end) //while there is a partial string matched to email pattern
        {
            numofemail++; //increase numberofemail
            iter2++; //find next match
        }
    }
    printf("%d\n%d\n", numofsecuritynum, numofemail); //print the number of personal information
}
```

Resulting output



4. Pro3

Brief Algorithm

```
make re representing personal information, security number and email
open input text file
while ( get a string line from input text file until there is no more string )
{
    while ( there is a partial string matched to personal information )
    {
        if ( partial string is matched to social security number)
            anonymize the social security number
        else if ( partial string is matched to email )
            anonymize the email

        print the anonymized information
        find next match
    }
}
```

For finding security number, regular expression "₩₩s((98|99|00|01)((0(1|3|5|7|8)|1(0|2))([0-2]₩₩d|30|31)|(0(4|6|9)|11)([0-2]₩₩d|30)|02([0-1]₩₩d|2[0-8]))|000229)-([1-4])[₩₩d]{6}₩₩s " is used.

For finding email, regular expression

"₩₩s([A-Za-z0-9]+)@([a-zA-Z]+)₩₩.ac₩₩.kr₩₩s" is used.

And for finding personal information (securitynumber + email), regular expression

"(₩₩s((98|99|00|01)((0(1|3|5|7|8)|1(0|2))([0-2]₩₩d|30|31)|(0(4|6|9)|11)([0-2]₩₩d|30)|02([0-1]₩₩d|2[0-8]))|000229)-([1-4])[₩₩d]{6}₩₩s)|(₩₩s([A-Za-z0-9]+)@([a-zA-Z]+)₩₩.ac₩₩.kr₩₩s)" is used.

An explanation of code is provided as annotations of following source code.

```
#include <stdio.h>
#include <regex>
#include <string>

using namespace std;

int main(int argc, char * argv[])
{
    regex re("(\\s((98|99|00|01)((0(1|3|5|7|8)|1(0|2))([0-2]\\d|30|31)|(0(4|6|9)|11)([0-
2]\\d|30)|02([0-1]\\d|2[0-8]))|000229)-([1-4])[\\d]{6}\\s)|(\\s([A-Za-z0-9]+)@([a-zA-
Z]+)\\.ac\\.kr\\s)");
```

```
    regex securitynum("\\s((98|99|00|01)((0(1|3|5|7|8)|1(0|2))([0-2]\\d|30|31)|(0(4|6|9)|11)([0-
2]\\d|30)|02([0-1]\\d|2[0-8]))|000229)-([1-4])[\\d]{6}\\s");
    regex email("\\s([A-Za-z0-9]+)@([a-zA-Z]+)\\.ac\\.kr\\s");

    FILE * fp1 = fopen(argv[1], "r"); //open the input file in the path specified by argument
    char s[1001]; //char array for input string
    const char * c;
    smatch match;
    string match_str;
    while (fgets(s, 1001, fp1) != NULL) //while there are strings to call in input file
    {
        string str = string(s);
        sregex_iterator iter(str.begin(), str.end(), re); //iterator to repeat matching string to pe
rsonal information pattern
        sregex_iterator end;

        while (iter != end) //while there is a partial string matched to personal information patter
n
        {
            match = *iter;
            match_str = match.str(); //convert to string
            if (regex_match(match_str, securitynum)) //if the partial string is matched to securityn
umber
                match_str = regex_replace(match_str, securitynum, "$1-
$12XXXXXX"); //anonymize security number
            else if (regex_match(match_str, email)) //else if the partial string is matched to email

                match_str = match.format("$15@XXXX.ac.kr"); //anonymize email

            c = match_str.c_str(); //convert to string
            printf("%s\n", c); //print anonymized information
            iter++; //find next match
        }
    }
}
```

Resulting output