ECE4574 FA23 – Prof. Jones – HW 3

Due Friday, November 10, 2023 – 11:59 PM via Canvas

In this assignment you will extend the code that you wrote in HW2 to satisfy some additional requirements. These are mostly non-functional requirements. Our software package is a great success and is being merged into a new product, but it has some architectural guidelines that we have to follow. All previous requirements are still in place, except for where they are obviously replaced by new ones. The new requirements are as follows:

1. Only a single OneTimePad object should exist at any time, even though multiple OTP coder objects may be running (they should all use the same pad). The single OneTimePad should still be regenerated by the UI and its current values displayed. Use a suitable design pattern to accomplish this.
2. Create new coder objects using a Factory pattern. The sample code below (from my main program) should be functional.
3. Create an abstract class that all coders inherit from.
4. I want to be able to chain coders together to form a new Composite coder. In other words, I would like to create a "coder" consisting of (for example) a flip coder and a base64 coder; input text would be flipped THEN base64 coded to produce the final output. Of course, decoding must be done in the reverse order. Use the Composite design pattern to accomplish this.
5. Add a data structure to your main class that contains all of the "coders" that you declare (in my code example below, I use a std::vector of coder component pointers). This way, you can just iterate through the data structure to run the coders.
6. Add a vertical scroll bar to the window in which your text output is displayed.
7. Once this is done, define and run suitable tests to demonstrate the following:
   - Two different input strings – no need to be too long
   - Two different OTPs
   - All single coders
   - At least one Composite coder consisting of an OTP coder and at least one other coder.
8. Draw a class diagram for your final project.

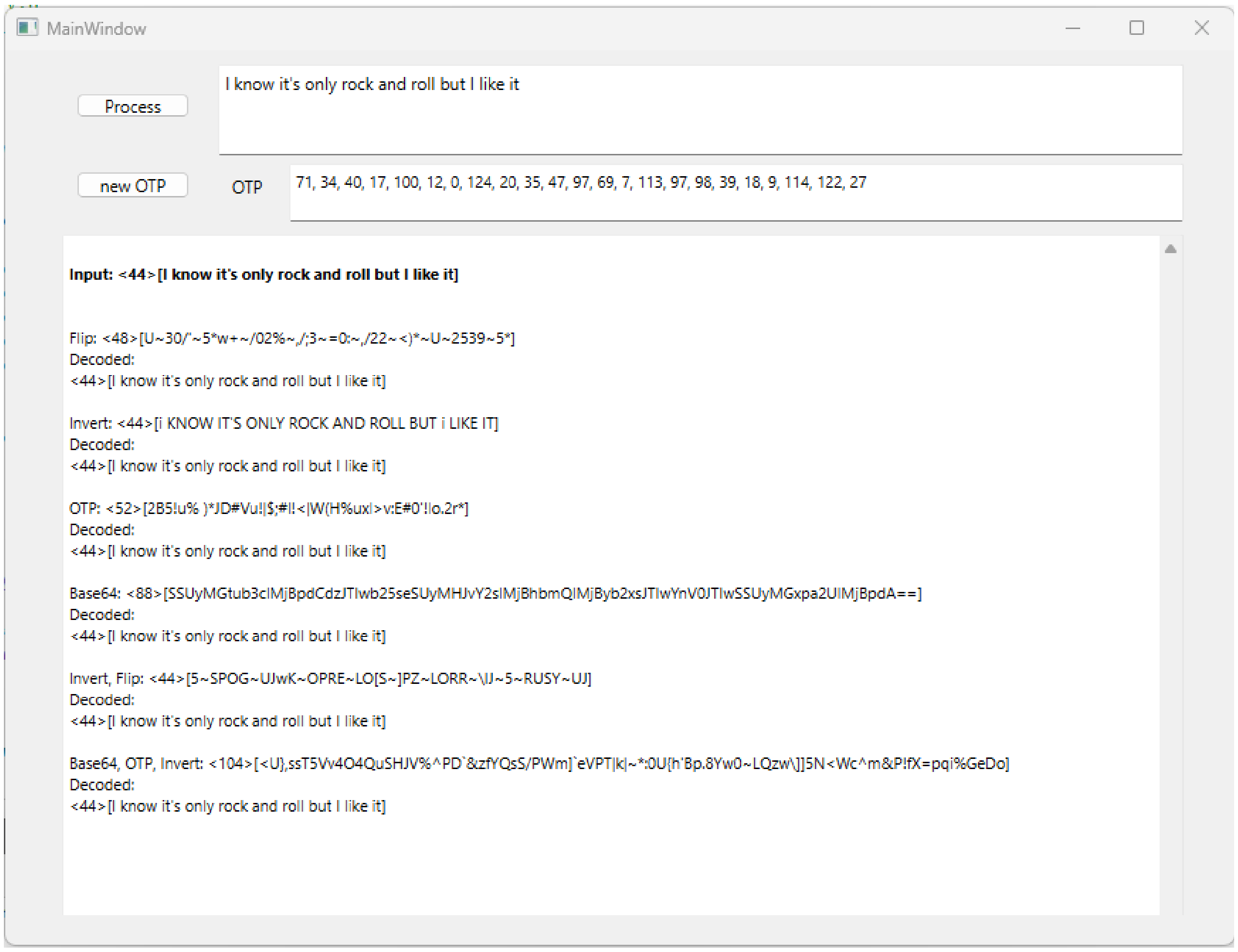
When your application is complete, submit:

- a single Word document, containing:
  - your class diagram
  - all of the code that you wrote pasted in as plain text (no screen shots or dark mode)

    - screen shots of your running application, similar to what you see below, demonstrating all of your test cases, saved as .png files!
- a single zip file containing all source files in your project .

Submit the Word document and zip file using Canvas.

Here is a set of sample output from my program…



```
I know it's only rock and roll but I like it
```

```
71, 34, 40, 17, 100, 12, 0, 124, 20, 35, 47, 97, 69, 7, 113, 97, 98, 39, 18, 9, 114, 122, 27
```

```
Input: <44>[I know it's only rock and roll but I like it]


Flip: <48>[U~30/'~5*w+~/02%~,/;3~=0:~,/22~<)*~U~2539~5*]
Decoded:
<44>[I know it's only rock and roll but I like it]

Invert: <44>[i KNOW IT'S ONLY ROCK AND ROLL BUT i LIKE IT]
Decoded:
<44>[I know it's only rock and roll but I like it]

OTP: <52>[2B5!u% )*JD#Vu!|$;#I!<|W(H%uxl>v:E#0'!lo.2r*]
Decoded:
<44>[I know it's only rock and roll but I like it]

Base64: <88>[SSUyMGtub3clMjBpdCdzJTIwb25seSUyMHJvY2slMjBhbmQlMjByb2xsJTIwYnV0JTIwSSUyMGxpa2UlMjBpdA==]
Decoded:
<44>[I know it's only rock and roll but I like it]

Invert, Flip: <44>[5~SPOG~UJwK~OPRE~LO[S~]PZ~LORR~\IJ~5~RUSY~UJ]
Decoded:
<44>[I know it's only rock and roll but I like it]

Base64, OTP, Invert: <104>[<U},ssT5Vv4O4QuSHJV%^PD`&zfYQsS/PWm]`eVPT|k|~*:0U{h'Bp.8Yw0~LQzw\]]5N<Wc^m&P!fX=pqi%GeDo]
Decoded:
<44>[I know it's only rock and roll but I like it]
```

And here is some example code from my main, showing how component creation should work (CoderComponent contains an enumeration with members FLIP, INVERTCASE, etc.):

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    CoderFactory cFactory;

    this->coders.push_back(cFactory.create(CoderComponent::FLIP));
    this->coders.push_back(cFactory.create(CoderComponent::INVERTCASE));
    this->coders.push_back(cFactory.create(CoderComponent::OTP));
    this->coders.push_back(cFactory.create(CoderComponent::BASE64));
    this->coders.push_back((cFactory.create(CoderComponent::COMPOSITE)));
    this->coders.back()->add(cFactory.create(CoderComponent::INVERTCASE));
    this->coders.back()->add(cFactory.create(CoderComponent::FLIP));
    this->coders.push_back((cFactory.create(CoderComponent::COMPOSITE)));
    this->coders.back()->add(cFactory.create(CoderComponent::BASE64));
    this->coders.back()->add(cFactory.create(CoderComponent::OTP));
    this->coders.back()->add(cFactory.create(CoderComponent::INVERTCASE));

    QString initstr = QString("I know it's only rock and roll but I like it");
<<OTHER STUFF HERE>>
}

void MainWindow::doProcess()
{
    // Clear the output box
    this->ui->outputTextBox->clear();

    QString inputstr, codedstr, decodedstr;

    // 1.      Get the input string from the inputTextBox. When you write the string out, first
write the total length of the string in decimal, surrounded by angle brackets, then write the
string surrounded by square brackets. So, if the string “Hello, world!” is the input, you
should write out: <13>[Hello, world!]
    printf("Enter the string to be encoded: ");
    inputstr = this->ui->inputTextBox->toPlainText();
    widgetWriteHTMLString(this->ui->outputTextBox, inputstr, "<b>Input:", "</b><br>");

    for (CoderComponent* pCoder : this->coders)
    {
        codedstr = pCoder->encode(inputstr);
        widgetWriteHTMLString(this->ui->outputTextBox, codedstr, QString("<br>") + pCoder-
>getType() + ":");
        decodedstr = pCoder->decode(codedstr);
        widgetWriteHTMLString(this->ui->outputTextBox, decodedstr, "Decoded:" "<br>");

    }
}
```