ECE4574 FA23 – Prof. Jones – HW 2

Due Friday, October 13, 2023 – 11:59 PM via Canvas

In this assignment you will extend the code that you wrote in HW1 to satisfy some additional requirements. The new requirements are as follows:

1. Create your project as a Qt Widget application. This will have a mainwindow UI (see below). The application should not terminate when it does one series of encodings, but should allow me to repeatedly enter text and encode/decode it.
2. Add a suitable, simple user interface to your project. At a minimum, I would like the following:
    - Within a single window, add a button (or other input device) to run the list of coders defined in your code. The text input should be run through the flip coder, then the output should be decoded, then run the input through the OTP coder and decode that result, then the same for the invert coder and the Base64 coder (see below).
    - Add a text box to allow the user to type the text to be encoded. Make the default text in this box: "THESE are the times that try men's souls."
    - Add a button (or other input device) to regenerate the one-time pad used by the OTPCoder, and a text box to display the current one-time pad. When the one-time pad is regenerated, compute a new list of random integers between 0 and 126 to use as the offsets; the length of the pad should be a random number between 16 and 32 (does not need to be a prime number).
    - When the coders are run, display the results in a simple text box as shown below. The input string should be displayed in **bold**, as I have done.
    - Finally, make good design choices for the UI – reasonable layout, etc. My example below is just an example – feel free to improve it.
3. As mentioned above, allow the one-time pad to be regenerated. When the regen button is pressed, a new pad is calculated and displayed. It is then used in all OTP calculations until the regen button is pressed again.
4. Add a base64 coder, that uses a web service to do the base64 coding. You will find information on the API that I want you to use at: https://networkcalc.com/api/ . There is more detail that you will need to know, but to encode a string *mystring*, you send a REST request using the URL `https://networkcalc.com/api/encoder/`*`mystring`*`?encoding=base64`

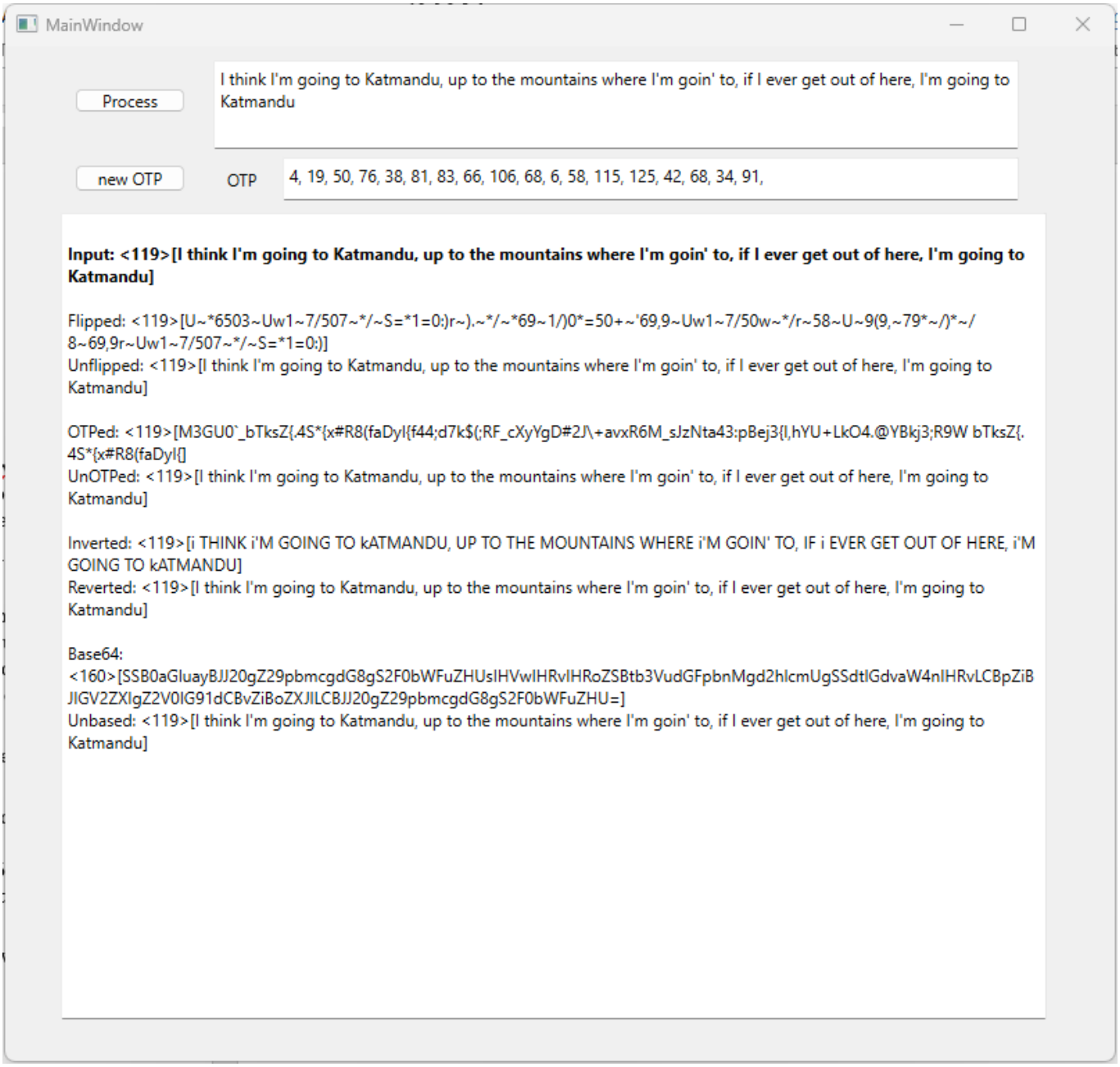
When your application is complete, submit:

- a single Word document, containing all of the code that you wrote pasted in as plain text (no screen shots or dark mode)
- a screen shot of your running application, similar to what you see below, saved as a .png file! Test your code on two strings (and be SURE to demonstrate two different OTPs):

    - “In a hole in the ground there lived a Hobbit 1234567890 ~!@#$%^&*()+=”
    - “England expects every man will do his duty”
- a single zip file containing:
- all source files in your project .

Submit the Word document, image file and zip file using Canvas.

Here is a set of sample output from my program…



One of the things that your base64 encoder will have to do is to wait until the remote REST server replies. This will require you to use Qt events – which we will be talking about in class, but here is a clue.

In my encode function, I need a GET request to the remote host. I use a separate class to do this; it's called RESTTalker (I showed you an example of a similar thing in the lecture on Web services). My Base64Coder class contains a member of type RESTTalker*. The RESTTalker object formats and actually sends the GET message using Qt network services. When it gets a reply, it emits a signal saying replyReceived. The code below waits for this replyReceived signal, and then calls a method of RESTTalker to get the result that was received (that function also has to pull out the part that I care about).

```
QString Base64Coder::encode(const QString in)
{
    QString fixedIn = QString(in);
    fixedIn.replace(QString(" "), QString("%20"));
    /* < your code here to create the proper urlstring > */
    QEventLoop loop;
    connect(this->pRESTTalker, &RESTTalker::replyReceived,
                &loop, &QEventLoop::quit);
    this->pRESTTalker->tryREST(urlstring);
    loop.exec(); //exec will delay execution until the signal has arrived
    this->result = this->parseResponse(this->pRESTTalker->getResponse());
    return this->result;
}
```