Departamento de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de Concepción
Concepción, Chile.

Informe N°6

Máquina Enigma

Profesor: Mario Medina

Alumno: Aldo Mellado Opazo

19 de diciembre de 2018

Programación Orientada al Objeto
Ingeniería Civil en Telecomunicaciones

1. Sobre el Código Enigma

Dado que el problema se presenta como una máquina cuyo objetivo era el de que, mediante el uso de 5 rotore distintos, de los cuales el usuario escogía 3, seleccionando además las letras con las cuales empezaba cada rotor, se debía presentar un menú a través del cual el usuario escogiera.

Paso siguiente, el problema pide que la codificación (o decodificación), se haga a través del paso del texto, caracter por caracter, por los tres rotores, de los cuales ya identificamos como rotor rápido, medio y lento. Luego de esto, pasaba por el reflector, y entonces, por el rotor lento, medio y rápido respectivamente.

Se debían hacer salvedades tales como que el rotor lento rotara 1 vez por cada 676 que hacía el rotor rápido, a la vez que el rotor mediano rotaba 1 vez cada 26 que hacía el rápido.

1.1. Problemas

Por practicidad, evaluaré los problemas de manera tal que abordaré los problemas presentados al comienzo, hasta llegar a los del final. Habiendo dicho esto,

1. `while(flag!=1):` Dado que el usuario debe elegir los rotores a usar, existe también la posibilidad de que el usuario escoja, erronea o intencionalmente un mismo rotor dos veces, por ello, debía considerarse dicha salvedad para el número de los rotores, así como para la letra desde la que se rota el rotor.

2. `convertir()`: El problema pedía que esta fuese una función miembro de la clase Enigma, sin embargo, debido al problema que se detalla en el punto 2, no fue posible.

3. `string avanzaRotor():` El problema dentro de esto, es que la función avanzaRotor, presentada a continuación:

```
1
2  string Rotor::avanzarRotor()
3  {
4      list<char> aux;
5      string aux1;
6      string::iterator iter = clave.begin();
7
8      for(iter;iter!=clave.end();iter++)
9      {
10         if(iter!=clave.end())
11         {
12             aux.push_front(*iter);
13         }
14         else
15         {
16             aux.push_back(*clave.begin());
17         }
18     }
19
20     clave.clear();
21
22     for(auto x:aux)
23     {
24         clave.push_back(x);
25     }
26     return clave;
27 }
```

   En la forma en que la pensé originalmente, era que entregara la clave del rotor avanzada, y así, que esta fuera introducida en un nuevo rotor que sería el rotor de la clave avanzada. Sin embargo, luego se me hizo el alcance de que en realidad lo que debería retornar era un rotor con la clave rotada en vez de la clave rotada.

   Esto ciertamente trajo problemas a la hora de avanzar el rotor conforme se introducían los caracteres y se verificaban las condiciones; `crr%26` y `crr%676`. Dicho problema se manifesto mediante no poder reutilizar

el rotor con la clave rotada.

Esto a su vez ocasionó que no pudiera rotarse el rotor cuando, luego de pasar por el reflector, siguiera haciendo el conteo, que según entendí, debía seguir avanzando conforme los caracteres pasaban por los rotores.

4. `encripta(const string& p)`: El problema de esta función, fue que al comienzo, la pensé para que encriptara palabra por palabra, sin embargo, dado que la función más adelante serviría para codificar letra por letra, debía entonces ser repensada y reacondicionada para recibir `char`

5. `string avanzaClave()`: También una cosa que se pensó fue hacer avanzar la clave, retornar una clave rotada y luego, introducirla a una función que retornara un rotor nuevo, que usara la clave del rotor anterior, pero esta vez rotada, sin embargo, no se logró retornar un rotor.

6. **Sobre el abecedario y caracteres especiales:** Se tiene que el texto a codificar, dada que se trata de uno de narrativa, y no de simples números o simples letras, cuenta además con caracteres especiales, signos de interrogación, exclamación y guiones que le dan coherencia al texto, que sí o sí debían traspasarse. Por lo que debía hacerse la salvedad para con estos.

1.2. Soluciones

1. `while(flag!=1):` A modo de evitar que el usuario pudiera cometer este error, propuse que evaluase los valores introducidos de la siguiente manera:

```
    while(flag!=1)
    {
        cout<<"\nFavor, escoja el numero de los rotores a usar: ";
        cin>>n1errotor>>n2orotor>>n3errotor;

        if(n1errotor!='1' && n1errotor!='2' && n1errotor!='3' && n1errotor!='4' && n1errotor!='5' ||
            n2orotor!='1' && n2orotor !='2'  && n2orotor!='3' && n2orotor!='4'  && n2orotor!='5'  ||
            n3errotor!='1'&& n3errotor!='2' && n3errotor!='3' && n3errotor!='4' && n3errotor!='5')
        {
            cout<<"\nUno de los nÃºmeros de rotor, no se corresponde a la cantidad de rotores
    disponibles, intente nuevamente"<<endl;
            flag=0;
        }
        else
        {
            if(n1errotor==n2orotor || n1errotor==n3errotor || n2orotor==n3errotor
            || n2orotor==n1errotor || n3errotor==n1errotor || n3errotor==n2orotor)
            {
                cout<<"\nExisten elementos Rotor repetidos, favor escoja nuevamente los nÃºmeros de
    rotor a usar"<<endl;
                flag=0;
            }
            else
            {
                flag=1;
                while(flag1!=1)
                {
                    cout<<"Escoja la primera letra de la clave a utilizar para cada rotor: ";
                    cin>>letra1>>letra2>>letra3;

                    if(isalpha(letra1)&&isalpha(letra2)&& isalpha(letra3))
                    {
                        if(letra1==letra2 || letra1==letra3 || letra2==letra3
                        || letra2==letra1 || letra3==letra1 || letra3==letra2)
                        {
                            cout<<"\n\t Existe una letra repetida, favor escoja nuevamente las letras a
     usar"<<endl;
                            flag1=0;
                        }
                        else
                        {
                            flag1=1;
                        }
                    }
                    else
                    {
                        cout<<"\nUno de los valores ingresados no es una letra, favor escoja nuevamente
     las letras a usar"<<endl;
                        flag=0;
                    }
                }

            flag=1;
            }
        }
    }
```

   De este modo, tanto si introduce el mismo número de rotor, así como de letras, o bien, si decide ingresar caracteres especiales en el número de rotores o en las letras con que se inicializa el rotor, el usuario podrá enmendar su error y optar a codificar el texto.

2. `convertir()`: Ciertamente la solución a convertir pudo haber sido el declararla como función miembro de Enigma y que esta hiciera las de desplegar el texto donde se señala que los rotores escogidos fueron los ingresados, que la codificación pasó por los rotores rapido, medio, lento, reflector, lento, medio y rapido respectivamente.

3. `char encripta(const char& p):` Para solucionar el problema, se reacondicionaron, tanto el funcionamiento de la función, así como los parametros que este recibe y retorna, de modo que resultó lo siguiente:

```
char Rotor::encripta(const char& p)
{
    char aux,salida,mensaje = p; //a
    aux = toupper(mensaje); // A

    map<char,char>::iterator iter2 = codificador.begin();

    if(codificador.find(aux)==codificador.end()) // Hace un push_back de los caracteres no mapeados en rotor e.g: ! , - , etc
        {
            salida  = aux;
        }
    else
        {
            salida = codificador.find(aux)->second; // busca la traduccion equivalente segun mapeo de rotor AA = QQ
        }
    return salida;
}
```

4. `string avanzaClave():` Para hacer avanzar la clave se pensó hacer uso de las funciones get(), and set(), donde estas fueran funciones miembro de la clase Enigma, sin embargo, por desconocimiento del uso.

5. **Sobre el abecedario y caracteres especiales:** Se implementó la siguiente solución al problema señalado.

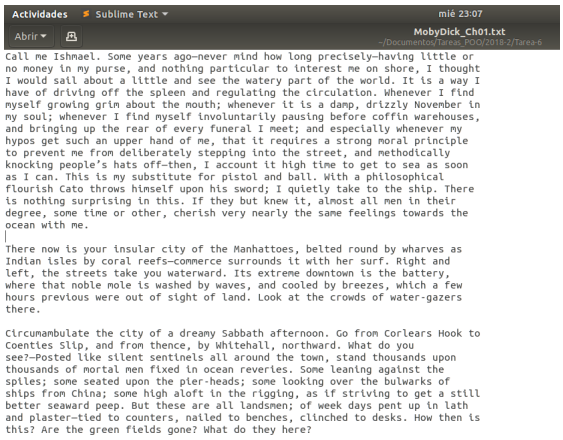
```
    if(codificador.find(aux)==codificador.end()) // Hace un push_back de los caracteres no mapeados en rotor e.g: ! , - , etc
        {
            salida  = aux;
        }
    else
        {
            salida = codificador.find(aux)->second; // busca la traduccion equivalente segun mapeo de rotor AA = QQ
        }
```

En ella se aprecia que mediante el uso de la función `.find()`, se hacía que, por funcionamiento de esta, si dentro de `codificador`, no se hallaba ningún elemento que coincidiera con los que este contenía, retornaba una posición al final de este, esto se ve en la línea:

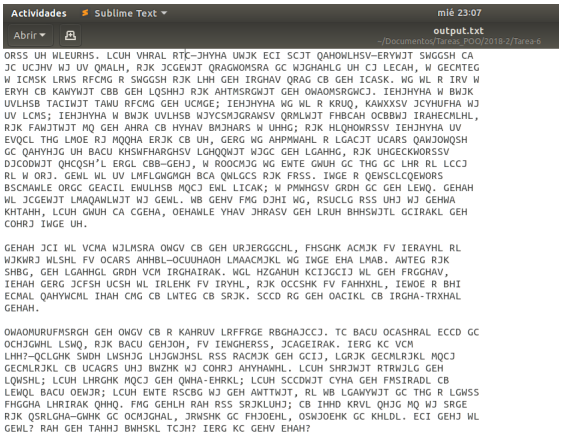
`if(codificador.find(aux)==codificador.end())`

Finalmente, si es que encontraba el caracter deseado, lo retornaba a salida, apuntando la equivalencia del símbolo encontrado a un char llamado `salida`.

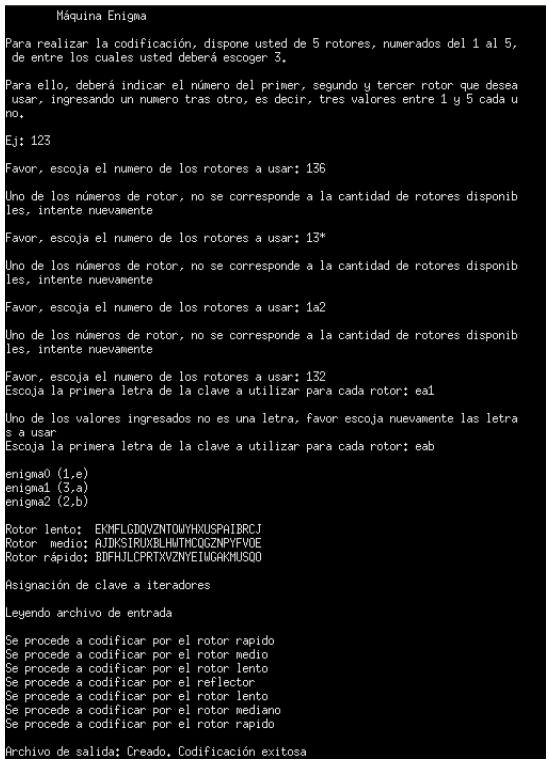
1.3. Resultados



(a) Texto original



(b) Texto encriptado



(c) Cuadro de diálogo desplegado