
Universidad de Costa Rica

IE-0217 - Estructuras abstractas de datos y algoritmos para ingeniería

Grupo de Trabajo #3

Profesor: M.Sc. Ricardo Roman Brenes

# Laboratorio 1: C++

*Estudiantes:* — *Carné:*
Natanael Mojica — A93901
Moisés Campos Zepeda — B31400

I semestre - 2017

# Índice

**1. Enunciado** **1**

**2. Solución** **2**
2.1. FileUtil . . . . . 2
2.2. Traslator . . . . . 3
2.3. Función main . . . . . 4

**3. Conclusiones** **6**

**4. Anexos** **7**
4.1. Clase Traslator . . . . . 7

# 1. Enunciado

Escriba y documente (con Doxygen) un programa, usando orientación a objetos, que genere los aminoácidos a partir de los codones de un código genético (ARN) cualquiera. Para esto, el programa recibe de la línea de comandos dos rutas, el archivo de entrada y el archivo de salida. El archivo de entrada estará compuesto por una serie de bases nitrogenadas (A, G, C, U) múltiplo de 3, que siempre empieza y termina por un codón de parada. Con este fin, se deben crear al menos dos clases Translator y FileUtil.

# 2. Solución

En la propuesta de laboratorio se indica una estructura a seguir para resolver el problema indicado. Basados en esta estructura se implementaron dos clases, cada una de ellas con un papel específico en la solución del problema:

- **FileUtil** es la clase encargada de realizar la lectura y escritura de archivos. Esta clase se encargará de leer el fichero que contiene la cadena de código genético ARN que se quiere convertir y creará un archivo que contendrá la cadena de aminoácidos resultado de la conversión.
- **Traslator** es la clase encargada de llevar a cabo la conversión de cadenas ARN a aminoácidos.

## 2.1. FileUtil

La clase lee el archivo de ejemplo con los codones mediante su función **read()** y los guarda en un string a través de **readLines()** para ser procesado por ***Translate***. La clase FileUtil, posee un método sobrecargados **write()** el cual en su primera definición acepta strings, y en su segunda punteros de string. Posee un constructor, el cual recibe las direcciones de los archivos y un destructor.

Listing 1: métodos read, readlines, y "write"s sobrecargados

```
1  string FileUtil::read(){
2
3    infile.open ( iFileName.c_str() );
4
5  }
6
7  string* FileUtil::readLines(){
8
9      while( getline( infile, input_data ) ) ///< leemos todo el archivo
10     {
11     out_data += input_data;
12     }
13   infile.close();
14   return new string(out_data);
15
16 }
17
18 int FileUtil::write( string s ){
19
20   outfile.open (oFileName);
21   cout << "escribiendo " << s << endl;
22     outfile << s << endl;
23     outfile << "\n";
24     outfile.close();
25     return 0;
26 }
27
28 int FileUtil::write( string *s ){
29
30   outfile.open (oFileName);
31     outfile << *s << endl;
32     cout << "escribiendo " << *s << endl;
33     outfile.close();
```

```
34      delete s;
35      return 0;
36
37 }
```

## 2.2. Traslator

La clase traslator tiene dos métodos sobrecargados llamados *traslate* que implementan el mismo algoritmo, que es utilizado para realizar la conversión de codones a aminoácidos. Debido a que son cerca de 64 codones y a la complejidad de código que seria requerido para evaluar la cadena entrante utilizando condicionales **if/else**, se decidió implementar un vector de la clase *std::vector* el cual contiene pares de objetos *string*, cada codón y su equivalente en aminoácido formarían un par que estaría ubicado en la posición *i* del vector, de este modo, la conversión se limita a evaluar la cadena entrante pasada como argumento al método, seccionarla en sub-grupos de 3 caracteres y comparar este sub-grupo o *substring* con cada posición **n-iésima** del vector para el primer par, si el sub-string es un codon válido se procederá a asignar el segundo par del vector en la posición i(en la cual esta el codon equivalente a cada substring) al resultado(aminoácido).

Listing 2: método traslate

```
1 string Traslator::traslate( string s ){
2
3     cut = 0;
4     for ( int i=0; i < ( s.size()/3 ); i++ )
5     {
6     sub = s.substr ( cut, 3 );
7   /*una cadena debe siempre empezar con algún codon de parada, si no empieza
8      * con alguno de ellos se añade al resultado
9      * */
10    if( ( cut == 0 ) && ( (sub != "UAA") || (sub != "UGA") || (sub != "UAG") ) ){
11      result += "|";
12    }
13    cout << "checking for: " << sub << endl;
14
15    for( int i=0; i < token.size(); i++ ){
16      if( sub.compare( token[i].first ) == 0 ){
17        result += token[i].second;
18        cout << sub << " is a token - parsing" << endl;
19        break;
20      }
21    }
22    cut = cut + 3;
23  }
24
25  /*ultimo caracter de la cadena debe ser "|", en caso contrario
26   * se añade
27   * */
28  if(result.substr(result.size()-1) != "|" ) result += "|" ;
29
30  return result;
31 }
```

Para implementar el algoritmo mencionado lineas arriba, se utilizaron dos ciclos **for**, el primeo itera sobre la longitud de la cadena de codones recibida como argumento(linea 4 de listings 2), debido

a que el argumento es un objeto del tipo string, es posible invocar a sus métodos *size()* y *substr()*, este último permite asignar a otro string un sub-string cuya longitud es posible especificar. En el segundo ciclo *for* se itera sobre cada posición del vector de modo que en cada posición se compara el substring obtenido del ciclo anterior con el codon encontrada en el primer par de cada posición del vector, si ambas cadenas son iguales el método *sub.compare(token[i].first* retornará cero, asignándose el segundo par del vector al string resultado(linea 17). Una vez concluido el primer ciclo for, se retorna el string resultado.

## 2.3. Función main

La función *main* recibe dos argumentos los cuales son pasados al programa ya compilado en tiempo de ejecución, estos parámetros le indican al programa donde buscar el archivo de entrada y donde guardar el archivo de salida con el resultado de la conversión. También se encarga de crear los nuevos objetos FileUtil y Traslator y hacer que ambos objetos interactuen a través de variables creadas en esta función, a las cuales se les asigna el valor retornado por los métodos invocados de cada objeto.

Listing 3: método main

```
1 #include "../include/fileutil.h"
2 #include "../include/traslator.h"
3
4 using namespace std;
5
6 int main(int argc, char *argv[])
7 {
8   cout << argc << endl;
9   if( argc < 3 ){
10     cout << "\033[1;31m ERROR \033[0m\n" << endl;
11     cout << "debe pasar dos argumentos: archivo de entrada y archivo de salida"
       << endl;
12     cout << "saliendo del programa " << endl;
13     return -1;
14   }
15
16     FileUtil *file = new FileUtil(argv[1], argv[2]);
17   Traslator *tras = new Traslator();
18
19     int i;
20   file->read();
21     string *patron =  file->readLines();
22
23     cout << "scanning: " << *patron << endl;
24   string *result = tras->traslate( patron );
25   cout << "ahora pasmos patron por valor: " << endl;
26
27   /* descomentar si se quiere verificar el método sobrecargado para
28    *  un argumento pasado por valor
29    * */
30
31   //string result2 = tras->traslate( *patron );
32
33   file->write( result ); //paso por puntero
34
```

```
35   /* descomentar si se quiere verificar el método sobrecargado para
36    *  un argumento pasado por valor
37    * */
38
39   //file->write( result2 ); //paso por valor
40
41   delete file;
42   delete tras;
43     return 0;
44 }
```

En las lineas 16 y 17, se crean dos nuevos objetos **file** y **tras**. El objeto file recibe como argumentos en su constructor el path de los archivos a leer y guardar, en cambio el objeto *tras* no recibe parámetros en su constructor. En la linea 20 se llama al método read del objeto file, luego se llama a su método *readLines* cuyo retorno es asignado a la variable patrón el cual es un puntero a string. Esta variable es pasada como argumento al método *traslate* del objeto *tras*, cuto retorno es el resultado de la conversión. Esta ultima variable será el argumento del método *write* del objeto *file*. Por último se elimina de la memoria ambos objetos y finaliza el programa

# 3. Conclusiones

- Implementando una lectura de un archivo de entrada y escritura en otro archivo, se logra que el algoritmo realice la conversión ARN a aminoácido, razón por la cual se puede transformar el string de entrada del archivo, en otro, a través del guardado en un string.
- Declarando un vector para realizar pares de objetos string, con codón y aminoácidos en la misma cadena a evaluar. Finalmente el main nos permite a través de sus parámetros darnos el resultado de la conversión.

# 4. Anexos

A continuación se presenta el código del trabajo:

## 4.1. Clase Traslator

Listing 4: Definición de la clase Traslator

```
#include <iostream>


class Traslator
{
  public:

    Traslator();
    ~Traslator();
    std::string traslate(std::string s);
    std::string* traslate(std::string *s);

    std::string result;


  private:

    int cut;
    std::string sub;


};
```

Listing 5: Implementación de la clase Traslator

```
1
2 #include <iostream>
3 #include <string>
4 #include <iostream>
5
6
7 #include "../include/traslator.h"
8 #include "../include/TablaVectores.h"
9
10 using namespace std;
11
12 Traslator::Traslator(){
13
14   result = "";
15   sub = "";
16   cut = 0;
17 }
18
19 Traslator::~Traslator(){
20
21 }
22
23 string Traslator::traslate( string s ){
24
25     cut = 0;
26     for ( int i=0; i < ( s.size()/3 ); i++ )
27     {
28     sub = s.substr ( cut, 3 );
29     /*una cadena debe siempre empezar con lgún codon de parada, si no empieza
30      * con algno de ellos se añade al resultado
31      * */
32     if( ( cut == 0 ) && ( (sub != "UAA") || (sub != "UGA") || (sub != "UAG") ) ){
33       result += "|";
34     }
35     cout << "checking for: " << sub << endl;
36
37     for( int i=0; i < token.size(); i++ ){
38       if( sub.compare( token[i].first ) == 0 ){
39         result += token[i].second;
40         cout << sub << " is a token - parsing" << endl;
41         break;
42       }
43     }
44     cut = cut + 3;
45   }
46
47   /*ultimo caracter de la cadena debe ser "|", en caso contrario
48    * se añade
49    * */
50   if(result.substr(result.size()-1) != "|" ) result += "|" ;
51
52   return result;
53 }
54
```

```
55 string* Traslator::traslate( string *s ){
56
57     cut = 0;
58     for ( int i=0; i < ( s->size()/3 ); i++ )
59     {
60     sub = s->substr ( cut, 3 );
61     /*una cadena debe siempre empezar con lgún codon de parada, si no empieza
62      * con algno de ellos se añade al resultado
63      * */
64     if( ( cut == 0 ) && ( (sub != "UAA") || (sub != "UGA") || (sub != "UAG") ) ){
65       result += "|";
66     }
67
68     cout << "checking for: " << sub << endl;
69     for( int i=0; i < token.size(); i++ ){
70       if( sub.compare( token[i].first ) == 0 ){
71         result += token[i].second;
72         cout << sub << " is a token - parsing" << endl;
73         break;
74       }
75     }
76     cut = cut + 3;
77   }
78   /*ultimo caracter de la cadena debe ser "|", en caso contrario
79    * se añade
80    * */
81   if(result.substr(result.size()-1) != "|" ) result += "|" ;
82
83   return new string(result);
84 }
```

Listing 6: Declaración y definición del vector de pares codon-aminoácido

```
1 #include <vector>
2 #include <iostream>
3
4 std::vector< std::pair <std::string, std::string> > token =
5 {
6 {"UUU","F"}, {"UUC","F"}, {"UUA","L"}, {"UUG", "L"}, {"CUU","L"}, {"CUC","L"}, {"
    CUA","L"},{"CUG","L"},
7 {"AUU","I"}, {"AUC","I"}, {"AUA","I"}, {"AUG","M"}, {"GUU","V"}, {"GUC","V"}, {"
    GUA","V"}, {"GUG","V"},
8 {"UCU","S"}, {"UCC","S"}, {"UCA","S"}, {"UCG","S"}, {"CCU","P"}, {"CCC","P"}, {"
    CCA","P"}, {"CCG","P"},
9 {"ACU","T"}, {"ACC","T"}, {"ACA","T"}, {"ACG","T"}, {"GCU","A"}, {"GCC","A"}, {"
    GCA","A"}, {"GCG","A"},
10 {"UAU","Y"}, {"UAC","Y"}, {"UAA","|"}, {"UAG","|"}, {"CAU","H"}, {"CAC","H"}, {"
    CAA","Q"}, {"CAG","Q"},
11 {"AAU","N"}, {"AAC","N"}, {"AAA","K"}, {"AAG","K"}, {"GAU","D"}, {"GAC","D"}, {"
    GAA","E"}, {"GAG","E"},
12 {"UGU","C"}, {"UGC","C"}, {"UGA","|"}, {"UGG","W"}, {"CGU","R"}, {"CGC","R"}, {"
    CGA","R"}, {"CGG","R"},
13 {"AGU","S"}, {"AGC","S"}, {"AGA","R"}, {"AGG","R"}, {"GGU","G"}, {"GGC","G"}, {"
    GGA","G"}, {"GGG","G"}
14 };
```