Proyecto para prueba de Doxygen

Generated by Doxygen 1.8.13

Contents

1 Práctica 10. Ficheros. Documentación de programas. Doxygen.

Objetivos

Los objetivos de esta práctica son que el alumnado:

- Desarrolle programas sencillos en C++ que utilicen ficheros, así como todas las características del lenguaje estudiadas
- Aloje todo el código fuente de sus programas en repositorios privados de GitHub
- Sepa depurar sus programas usando la interfaz de depuración del VSC
- Conozca la herramienta Doxygen
- Incluya en sus programas comentarios en el formato requerido por Doxygen

Rúbrica de evaluacion de esta práctica

Se señalan a continuación los aspectos más relevantes (la lista no es exhaustiva) que se tendrán en cuenta a la hora de evaluar esta práctica:

- El alumnado ha de acreditar conocer los conceptos expuestos en el material de referencia de esta práctica.
- El alumnado ha de acreditar que ha realizado todos los ejercicios propuestos, así como ser capaz de desarrollar otros similares.
- Ha de acreditar que es capaz de escribir un fichero Makefile para automatizar el proceso de compilación de sus programas.
- El código que escriba ha de estar escrito de acuerdo a los estándares definidos en la Guía de Estilo de Google para C++.
- Todos los identificadores que utilice en su programa (constantes, variables, etc.) deberán ser siempre significativos. No utilice nunca identificadores de una única letra, tal vez con la excepción de las variables que utilice para iterar en un bucle.
- Antes de su ejecución, todos los programas que desarrolle, deben imprimir en pantalla un mensaje indicando la finalidad del programa así como la información que precisará del usuario para su correcta ejecución.
- Ante la presencia de cualquier bug, el alumnado ha de conocer las técnicas básicas de depuración usando VSC
- Todos los ficheros de código del proyecto correspondiente a esta práctica han de alojarse en un repositorio de GitHub
- Los programas deben contener comentarios adecuados en el formato requerido por Doxygen

Introducción a la criptografía

Durante la segunda guerra mundial, el ejército alemán utilizó la máquina conocida como *Enigma* para codificar sus mensajes. Básicamente dada una *semilla* la máquina generaba una secuencia de números pseudoaleatorios que era difícil de reproducir, incluso aunque los detalles técnicos de la máquina pudieran ser descubiertos. Los aliados habían capturado algunas de las máquinas *Enigma*, de modo que conocían la forma en que la máquina trabajaba, pero los trabajos que se realizaron para descubrir los códigos de *Enigma* fueron los fundamentos de la criptografía moderna. Alan Turing participó en este tipo de trabajos. Si está interesado en conocer más sobre esta historia vea la película `The Imitation Game` (*Descifrando Enigma* en español).

La criptología es la rama de conocimiento que se ocupa del estudio y diseño de sistemas que permiten comunicaciones secretas entre un emisor de un mensaje y uno o varios receptores del mismo. Inicialmente las únicas aplicaciones de la criptología fueron militares, pero hoy en día son muchísimas otras. Por ejemplo, en los ordenadores multiusuario, cada usuario mantiene sus ficheros de una forma que no sean legibles para otros usuarios "indiscretos". Para conseguir esto, los ficheros se codifican (encriptan) utilizando una clave que sólo conoce su propietario. Mucha de la información que enviamos a través de internet viaja también de forma codificada para protegerla de receptores no deseados.

Para encriptar un fichero hay muchas alternativas. Todas ellas consisten en transformar cada uno de los caracteres del fichero original en otro carácter diferente siguiendo una determinada transformación. Indicaremos dos métodos diferentes de encriptado.

Encriptado xor (or exclusiva)

El Cifrado XOR requiere una clave secreta de encriptado/desencriptado. A cada uno de los caracteres del fichero se le hará una transformación, que consistirá en hacerle la operación `xor` con un carácter de la clave secreta. Estudie el capítulo Bitwise operators del tutorial y recuerde que el operador (de bits) en C/C++ es `^`

El carácter de la clave secreta con el que se transforma el carácter original, se variará de forma cíclica. P. ej. si suponemos que la clave secreta es la palabra `alfa`, y las primeras palabras del texto son *Informática Básica* los primeros caracteres del fichero de salida serán:

```
carácter 1  I xor a
carácter 2  n xor l
carácter 3  f xor f
carácter 4  o xor a
carácter 5  r xor a
carácter 6  m xor l
carácter 7  á xor f
carácter 8  t xor a
carácter 9  i xor a
carácter 9  c xor l
carácter 10 a xor f
carácter 11   xor a
carácter 12 B xor a
carácter 13 á xor l
carácter 14 s xor f
   ...
```

Es decir, se va haciendo la operación `xor` de cada uno de los caracteres del texto de entrada con cada uno de los caracteres de la clave secreta, tomando la clave de forma cíclica (cuando se acaba con el último carácter de la clave, se comienza de nuevo con el primero).

Antes de operar de este modo se procesará la clave secreta haciendo `xor` a cada uno de sus caracteres con la constante 128.

Una ventaja de este método es su especial aptitud para ser utilizado en un ordenador puesto que la operación o exclusiva se realiza muy eficientemente en un ordenador. Otra ventaja del método es que la operación de desencriptado consiste en hacer exactamente lo mismo al texto que se ha encriptado (con la misma clave secreta, por supuesto).

Cifrado de César

Como se deduce de su nombre, este método era usado ya en tiempos de los romanos. En este caso, la codificación es como sigue: si una letra en el texto a codificar es la N-ésima letra del alfabeto, sustitúyase esa letra por la (N + K)-ésima letra del alfabeto. (César utilizaba el valor K = 3). Se muestra a continuación un texto encriptado siguiendo este método y utilizando K = 1:

```
Texto original:  Navidad, Navidad, dulce navidad
Texto encriptado:  Obwjebe-!Obwjebe-!evmdf!obwjebe
```

Se puede optar por hacer fijo el valor de K o bien solicitarlo al usuario.

Evidentemente, el desencriptado del fichero consistirá en realizar la operación inversa, y en este caso, el valor de K a utilizar debería solicitarse al usuario para garantizar que está autorizado a leer el fichero.

Entorno de trabajo

Haga que el proyecto correspondiente a esta práctica conste al menos de 3 ficheros:

- Un fichero `cripto.cc` (programa principal) que contendrá la función `main` e incluirá el fichero de cabecera `funciones_cripto.h`
- El fichero `funciones_cripto.h` que contendrá las declaraciones de las diferentes funciones que se utilizan en el programa principal.
- El fichero `funciones_cripto.cc` que contendrá el código (definiciones) de las funciones declaradas en el fichero de cabecera.

La compilación del programa ha de estar automatizada mediante un fichero Makefile.

Desarrolle su programa de forma incremental, probando cada una de las funciones que va Ud. desarrollando. Utilice el depurador de VSC para corregir cualquier tipo de error semántico que se produzca en cualquiera de sus desarrollos.

Ejercicio

1. Desarrolle en C++ un programa `cripto.cc` cuya finalidad será encriptar y/o desencriptar ficheros de texto. Si el programa se ejecuta sin pasar parámetros en la línea de comandos, debemos obtener el siguiente mensaje:

```
./cripto -- Cifrado de ficheros
Modo de uso: ./cripto fichero_entrada fichero_salida método password operación
Pruebe ./cripto --help para más información
```

Si el programa se ejecuta pasando la opción `--help` se ha de obtener:

```
./cripto -- Cifrado de ficheros
Modo de uso: ./cripto fichero_entrada fichero_salida método password operación

fichero_entrada: Fichero a codificar
fichero_salida:  Fichero codificado
método:          Indica el método de encriptado
                   1: Cifrado xor
                   2: Cifrado de César
password:        Palabra secreta en el caso de método 1, Valor de K en el método 2
operación:       Operación a realizar en el fichero
                   +: encriptar el fichero
                   -: desencriptar el fichero
```

El programa solo se ejecutará cuando se le hayan pasado por línea de comandos los parámetros necesarios. En caso de detectar cualquier inconsistencia en los parámetros, el programa debe abortar su ejecución. Se indicará asimismo un mensaje de error si el programa no consigue abrir el fichero de entrada.

Documentación de código. Doxygen

Doxygen es una herramienta de código abierto que permite generar documentación de referencia para proyectos de desarrollo software. La documentación está escrita en el propio código fuente de los programas, y por lo tanto es relativamente fácil de mantener actualizada. Doxygen puede hacer referencias cruzadas entre la documentación y el código, de modo que el lector de un documento puede referirse fácilmente al código fuente. La herramienta extrae la documentación de los comentarios de los ficheros de código fuente y puede generar la salida en diferentes formatos entre los cuales están HTML, PDF LaTeX o páginas man de Unix.

En esta asignatura no se propone un uso exhaustivo de Doxygen pero sí se promueve que la documentación de los programas desarrollados se realice en el formato reconocido por Doxygen, que se ha convertido en un estándar de facto.

Comience por instalar Doxygen en su máquina virtual de la asignatura:

```
$ sudo apt install doxygen
```

Instale también los siguientes paquetes:

```
$ sudo apt install texlive-latex-base
$ sudo apt install texlive-latex-recommended
$ sudo apt install texlive-latex-extra
```

Estos paquetes son necesarios para compilar ficheros en formato Latex. Más adelante en este documento se justifica la necesidad de los programas que suministran estos paquetes.

En el manual de Doxygen indica cómo comenzar a trabajar con la herramienta. Si, ubicados en un directorio de trabajo se invoca `doxygen`:

```
doxygen -g <config-file>
```

la herramienta creará un fichero de configuración. Si no se le pasa el nombre del fichero como parámetro, creará un fichero con nombre `Doxyfile` preconfigurado para su uso. En el directorio de trabajo de esta práctica (`src`) se encuentra un fichero `Doxyfile` ya listo para usarse con proyectos de C++. Se ha incluído asimismo el código fuente de un programa para ilustrar con el mismo el uso de documentación con Doxygen. Si revisa Ud. el fichero `Doxyfile` (es un fichero de texto) verá toda una serie de opciones que el programa permite. Cada opción va precedida de una explicación de su finalidad y funcionamiento, de modo que puede Ud. probar a modificar algunas de ellas si lo desea. En esta página puede consultarse la finalidad y funcionamiento de cada una de las etiquetas (Tags) que se usan en el fichero de configuración de Doxygen.

Para generar la documentación de su aplicación, colóquese en el directorio de su proyecto (`src` en esta práctica) y ejecute:

```
doxygen Doxyfile
```

Con el fichero `Doxyfile` que se suministra, la herramienta creará un subdirectorio `doc` en el directorio raíz de su proyecto en el que alojará toda la documentación generada. El directorio donde Doxygen genera su salida se especifica con la etiqueta `OUTPUT_DIRECTORY` (línea 61 del fichero `Doxyfile` suministrado). Con la configuración suministrada se generan dos subdirectorios dentro de `doc`: `html` y `latex`. Si abre con un navegador el fichero `doc/html/index.html` accederá a la página principal de la documentación generada para el programa. Si se coloca en el directorio `doc/latex/` y ejecuta `make` el sistema "compila" el código latex y genera un fichero `refman.pdf` que contiene igualmente la documentación generada.

Tal como se ha indicado, HTML o latex son solo dos de los formatos que permite generar Doxygen. Tanto HTML como Latex (también Markdown) son lo que se conoce como lenguajes de marcas. HTML es el lenguaje que se utiliza para componer los textos que se muestran en las páginas web. Latex es un sistema de composición de textos que cuida el formato en especial en el ámbito de la tipografía y que es especialmente adecuado para textos de carácter científico. No se pretende aquí que profundice Ud. en conocer HTML o Latex.

La sección Documenting the code del manual de Doxygen indica cómo comentar el código fuente de modo que los comentarios sean procesados por Doxygen para incorporarlos a la documentación generada.

La guía Documenting C++ Code de documentación de código del proyecto LLST es la referencia que se adoptará en la asignatura para documentar el código de los programas que se desarrollen. Se utilizarán comentarios de tipo JavaDoc para comentarios de bloque:

```
/**
 * ... text ...
 */
```

JavaDoc es otro sistema de documentación ideado para Java y que también es muy popular. Doxygen soporta el uso de etiquetas "al estilo Javadoc" en el código.

Los bloques de comentarios multi-línea deben comenzar con

```
/**
```

y finalizar con

```
*/
```

Los comentarios de una única línea deben comenzar con ///. Por consistencia no use las opciones

```
/*!
```

o

```
//!
```

permitidas en Doxygen.

Así el bloque de comentarios que debe preceder a cualquier función (o método) tendrá la siguiente apariencia:

```
/**
 * Sum numbers in a vector.
 *
 * @param values Container whose values are summed.
 * @return sum of `values`, or 0.0 if `values` is empty.
 */
double sum(std::vector<double>& const values) {
  ...
}
```

En el ejemplo anterior @param y @return son etiquetas de tipo Javadoc. En Overview of supported JavaDoc style tags pueden consultarse este tipo de etiquetas.

El siguiente es un ejemplo (plantilla) de comentario de bloque que debería incluirse al comienzo de todos los ficheros (∗.cc, ∗.h) de un proyecto de programación en el ámbito de esta asignatura: “‘ /∗∗

- Universidad de La Laguna
- Escuela Superior de Ingeniería y Tecnología
- Grado en Ingeniería Informática
- Informática Básica
-
-

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 File Documentation

3.1 cripto.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include "cripto_funciones.h"
```

Include dependency graph for cripto.cc:

3.2 cripto_funciones.cc File Reference

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <iomanip>
#include "cripto_funciones.h"
```

Include dependency graph for cripto_funciones.cc:

Functions

- void Usage (int argc, char ∗argv[ ])

  Muestra el modo de uso correcto del programa En caso de que el uso no sea el correcto, muestra el mensaje y finaliza la ejecución del programa.
- void ComprobarParametros (char ∗argv[ ])

  Comprueba que los parámetros introducidos son los necesarios para el correcto proceso de encriptado, en caso contrario imprime en pantalla el mensaje de error correspondiente indicando que parámetro es incorrecto.
- bool IsInteger (std::string str)

  La siguiente función IsInteger comprueba si el parámetro k que se introduce por línea de comandos es realmente un número entero, tal y como se precisa.
- std::string EncriptadoXor (std::string contenido, std::string clave)

  A continuación, se presenta una función que encripta un fichero con el método de XOR.
- std::string EncriptadoCesar (std::string contenido, const int k)

  A continuación, se presenta una función que encripta un fichero con el método de césar.
- std::string DesencriptadoCesar (std::string contenido, const int k)

  A continuación, se presenta una función que desencripta un fichero previamente encriptado con el método de césar.

3.2.1 Function Documentation

3.2.1.1 ComprobarParametros()

```
void ComprobarParametros (
            char * argv[] )
```

Comprueba que los parámetros introducidos son los necesarios para el correcto proceso de encriptado, en caso contrario imprime en pantalla el mensaje de error correspondiente indicando que parámetro es incorrecto.

Definition at line 40 of file cripto_funciones.cc.

References IsInteger().

Referenced by main().

```
40                                      {
41   if (argv[3][0] != '1' && argv[3][0] != '2'){
42     std::cout << "Error en el método de encriptado" << std::endl;
43     exit(EXIT_FAILURE);
44   }
45   if (argv[3][0] == '2'){
46     std::string k{argv[4][0]};
47     if(IsInteger(k)){
48
49     }
50     else{
51       std::cout << "Error en el método de encriptado" << std::endl;
52       exit(EXIT_FAILURE);
53     }
54   }
55   if (argv[5][0] != '+' && argv[5][0] != '-'){
56     std::cout << "Error en la operación" << std::endl;
57     exit(EXIT_FAILURE);
58   }
59
60 }
```

Here is the call graph for this function: Here is the caller graph for this function:

3.2.1.2 DesencriptadoCesar()

```
std::string DesencriptadoCesar (
            std::string contenido,
            const int k )
```

A continuación, se presenta una función que desencripta un fichero previamente encriptado con el método de césar.

A la función se le pasa el contenido del fichero y la constate k.

Definition at line 112 of file cripto_funciones.cc.

Referenced by main().

```
112                                                                 {
113   std::string contenido_desencriptado;
114   for (int i = 0; i < contenido.length(); ++i){
115     char caracter_evaluar = contenido[i];
116     int caracter = caracter_evaluar;
117     caracter -= k;
118     char caracter_nuevo{caracter};
119     contenido_desencriptado += caracter_nuevo;
120   }
121   return contenido_desencriptado;
122 }
```

Here is the caller graph for this function:

3.2.1.3 EncriptadoCesar()

```
std::string EncriptadoCesar (
              std::string contenido,
              const int k )
```

A continuación, se presenta una función que encripta un fichero con el método de césar.

A la función se le pasa el contenido del fichero y la constate k.

Definition at line 96 of file cripto_funciones.cc.

Referenced by main().

```
96                                                          {
97   std::string contenido_encriptado;
98   for (int i = 0; i < contenido.length(); ++i){
99     char caracter_evaluar = contenido[i];
100     int caracter = caracter_evaluar;
101     caracter += k;
102     char caracter_nuevo{caracter};
103     contenido_encriptado += caracter_nuevo;
104   }
105   return contenido_encriptado;
106 }
```

Here is the caller graph for this function:

3.2.1.4 EncriptadoXor()

```
std::string EncriptadoXor (
              std::string contenido,
              std::string clave )
```

A continuación, se presenta una función que encripta un fichero con el método de XOR.

A la función se le pasa el contenido del fichero y la clave o contraseña correspondiente.

Definition at line 79 of file cripto_funciones.cc.

Referenced by main().

```
79                                                          {
80   std::string contenido_encriptado{""};
81   std::string clave_encriptada{""};
82   for (auto i : clave){
83     clave_encriptada += (clave[i]^128);
84     std::cout << clave_encriptada;
85   }
86   for (int i : contenido){
87     contenido_encriptado += (contenido[i]^clave_encriptada[i%clave_encriptada.length()]);
88   }
89   return contenido_encriptado;
90 }
```

Here is the caller graph for this function:

3.2.1.5 IsInteger()

```
bool IsInteger (
            std::string str )
```

La siguiente función IsInteger comprueba si el parámetro k que se introduce por línea de comandos es realmente un número entero, tal y como se precisa.

Definition at line 66 of file cripto_funciones.cc.

Referenced by ComprobarParametros().

```
66                                {
67   for(auto c: str){
68     if(c < 48 || c > 57)
69       return false;
70   }
71
72   return true;
73 }
```

Here is the caller graph for this function:

3.2.1.6 Usage()

```
void Usage (
            int argc,
            char * argv[] )
```

Muestra el modo de uso correcto del programa En caso de que el uso no sea el correcto, muestra el mensaje y finaliza la ejecución del programa.

Parameters

in	*argc*	Number of command line parameters
in	*argv*	Vector containing (char∗) the parameters

Definition at line 15 of file cripto_funciones.cc.

References kHelpText.

Referenced by main().

```
15                                    {
16
17   if (argc == 1 && argc != 6) {
18     std::cout << argv[0] << ": faltan parámetros" << std::endl;
19     std::cout << "Pruebe " << argv[0] << " --help para más información" << std::endl;
20     exit(EXIT_SUCCESS);
21   }
22   if (argc == 2){
23     std::string parameter{argv[1]};
24     if (parameter == "--help") {
25       std::cout << kHelpText << std::endl;
26       exit(EXIT_SUCCESS);
27     }
28   }
29   if (argc != 6){
30     std::cout << argv[0] << ": faltan parámetros" << std::endl;
31     std::cout << "Pruebe " << argv[0] << " --help para más información" << std::endl;
32     exit(EXIT_SUCCESS);
33   }
34 }
```

Here is the caller graph for this function:

3.3 cripto_funciones.h File Reference

```
#include <iostream>
#include <fstream>
#include <string>
```

Include dependency graph for cripto_funciones.h: This graph shows which files directly or indirectly include this file:

Functions

- void Usage (int argc, char *argv[])

  Muestra el modo de uso correcto del programa En caso de que el uso no sea el correcto, muestra el mensaje y finaliza la ejecución del programa.
- void ComprobarParametros (char *argv[])

  Comprueba que los parámetros introducidos son los necesarios para el correcto proceso de encriptado, en caso contrario imprime en pantalla el mensaje de error correspondiente indicando que parámetro es incorrecto.
- bool IsInteger (std::string str)

  La siguiente función IsInteger comprueba si el parámetro k que se introduce por línea de comandos es realmente un número entero, tal y como se precisa.
- std::string EncriptadoXor (std::string contenido, std::string clave)

  A continuación, se presenta una función que encripta un fichero con el método de XOR.
- std::string EncriptadoCesar (std::string contenido, const int k)

  A continuación, se presenta una función que encripta un fichero con el método de césar.
- std::string DesencriptadoCesar (std::string contenido, const int k)

  A continuación, se presenta una función que desencripta un fichero previamente encriptado con el método de césar.

Variables

- const std::string kHelpText

3.3.1 Function Documentation

3.3.1.1 ComprobarParametros()

```
void ComprobarParametros (
             char * argv[] )
```

Comprueba que los parámetros introducidos son los necesarios para el correcto proceso de encriptado, en caso contrario imprime en pantalla el mensaje de error correspondiente indicando que parámetro es incorrecto.

Definition at line 40 of file cripto_funciones.cc.

References IsInteger().

Referenced by main().

```
40                                      {
41   if (argv[3][0] != '1' && argv[3][0] != '2'){
42     std::cout << "Error en el método de encriptado" << std::endl;
43     exit(EXIT_FAILURE);
44   }
45   if (argv[3][0] == '2'){
46     std::string k{argv[4][0]};
47     if(IsInteger(k)){
48
49     }
50     else{
51       std::cout << "Error en el método de encriptado" << std::endl;
52       exit(EXIT_FAILURE);
53     }
54   }
55   if (argv[5][0] != '+' && argv[5][0] != '-'){
56     std::cout << "Error en la operación" << std::endl;
57     exit(EXIT_FAILURE);
58   }
59
60 }
```

Here is the call graph for this function: Here is the caller graph for this function:

3.3.1.2 DesencriptadoCesar()

```
std::string DesencriptadoCesar (
              std::string contenido,
              const int k )
```

A continuación, se presenta una función que desencripta un fichero previamente encriptado con el método de césar.

A la función se le pasa el contenido del fichero y la constate k.

Definition at line 112 of file cripto_funciones.cc.

Referenced by main().

```
112                                                                 {
113   std::string contenido_desencriptado;
114   for (int i = 0; i < contenido.length(); ++i){
115     char caracter_evaluar = contenido[i];
116     int caracter = caracter_evaluar;
117     caracter -= k;
118     char caracter_nuevo{caracter};
119     contenido_desencriptado += caracter_nuevo;
120   }
121   return contenido_desencriptado;
122 }
```

Here is the caller graph for this function:

3.3.1.3 EncriptadoCesar()

```
std::string EncriptadoCesar (
              std::string contenido,
              const int k )
```

A continuación, se presenta una función que encripta un fichero con el método de césar.

A la función se le pasa el contenido del fichero y la constate k.

Definition at line 96 of file cripto_funciones.cc.

Referenced by main().

```
96                                                          {
97   std::string contenido_encriptado;
98   for (int i = 0; i < contenido.length(); ++i){
99     char caracter_evaluar = contenido[i];
100     int caracter = caracter_evaluar;
101     caracter += k;
102     char caracter_nuevo{caracter};
103     contenido_encriptado += caracter_nuevo;
104   }
105   return contenido_encriptado;
106 }
```

Here is the caller graph for this function:

3.3.1.4 EncriptadoXor()

```
std::string EncriptadoXor (
              std::string contenido,
              std::string clave )
```

A continuación, se presenta una función que encripta un fichero con el método de XOR.

A la función se le pasa el contenido del fichero y la clave o contraseña correspondiente.

Definition at line 79 of file cripto_funciones.cc.

Referenced by main().

```
79                                                          {
80   std::string contenido_encriptado{""};
81   std::string clave_encriptada{""};
82   for (auto i : clave){
83     clave_encriptada += (clave[i]^128);
84     std::cout << clave_encriptada;
85   }
86   for (int i : contenido){
87     contenido_encriptado += (contenido[i]^clave_encriptada[i%clave_encriptada.length()]);
88   }
89   return contenido_encriptado;
90 }
```

Here is the caller graph for this function:

3.3.1.5 IsInteger()

```
bool IsInteger (
              std::string str )
```

La siguiente función IsInteger comprueba si el parámetro k que se introduce por línea de comandos es realmente un número entero, tal y como se precisa.

Definition at line 66 of file cripto_funciones.cc.

Referenced by ComprobarParametros().

```
66                              {
67   for(auto c: str){
68     if(c < 48 || c > 57)
69       return false;
70   }
71
72   return true;
73 }
```

Here is the caller graph for this function:

3.3.1.6 Usage()

```
void Usage (
              int argc,
              char * argv[] )
```

Muestra el modo de uso correcto del programa En caso de que el uso no sea el correcto, muestra el mensaje y finaliza la ejecución del programa.

Parameters

in	*argc*	Number of command line parameters
in	*argv*	Vector containing (char∗) the parameters

Definition at line 15 of file cripto_funciones.cc.

References kHelpText.

Referenced by main().

```
15                                  {
16
17   if (argc == 1 && argc != 6) {
18     std::cout << argv[0] << ": faltan parámetros" << std::endl;
19     std::cout << "Pruebe " << argv[0] << " --help para más información" << std::endl;
20     exit(EXIT_SUCCESS);
21   }
22   if (argc == 2){
23     std::string parameter{argv[1]};
24     if (parameter == "--help") {
25       std::cout << kHelpText << std::endl;
26       exit(EXIT_SUCCESS);
27     }
28   }
29   if (argc != 6){
30     std::cout << argv[0] << ": faltan parámetros" << std::endl;
31     std::cout << "Pruebe " << argv[0] << " --help para más información" << std::endl;
32     exit(EXIT_SUCCESS);
33   }
34 }
```

Here is the caller graph for this function:

3.3.2 Variable Documentation

3.3.2.1 kHelpText

```
const std::string kHelpText
```

Initial value:

```
= R"(fichero_entrada: Fichero a codificar
fichero_salida:  Fichero codificado
método:          Indica el método de encriptado
                   1: Cifrado xor
                   2: Cifrado de César
password:        Palabra secreta en el caso de método 1, Valor de K en el método 2
operación:       Operación a realizar en el fichero
                   +: encriptar el fichero
                   -: desencriptar el fichero)"
```

Definition at line 5 of file cripto_funciones.h.

Referenced by Usage().

3.4 Files-Doxygen.md File Reference

Index