Laboratorio 2024 de Programación 1 - S2

Tarea 2

Información general

Se sugiere leer con mucha atención todo el texto antes de comenzar la tarea. Es muy importante que se respeten todos los requisitos solicitados en esta sección y las siguientes. Si surgen dudas, pedimos que las formulen en el foro correspondiente a la tarea.

Individualidad

Esta tarea se deberá realizar de forma **individual**. Para todas las tareas rige el Reglamento del Instituto de Computación ante Instancias de No Individualidad en los Laboratorios (lectura obligatoria).

También está prohibido el uso de herramientas de inteligencia artificial (como ChatGPT, Bard, CoPilot, etc.) para generar código del laboratorio.

Calendario

Entrega: La entrega de la tarea puede realizarse **hasta las 20:00 del martes 19 de noviembre**. Los trabajos deberán ser entregados dentro de los plazos establecidos. **No** se aceptarán trabajos fuera de plazo. Para poder entregar tiene que haber realizado anteriormente el ***Cuestionario 6***.

Re-Entrega: Todos los estudiantes que realizaron la entrega pueden hacer modificaciones y realizar una segunda entrega (*re-entrega*). La re-entrega puede realizarse **hasta las 20:00 del jueves 21 de noviembre**.

Forma de entrega

Se debe entregar un **único** archivo de nombre **tarea2.pas** que debe contener **únicamente** el código de los subprogramas pedidos y eventualmente el de subprogramas auxiliares que se necesiten para implementarlos.

Archivos provistos y ejecución de pruebas

El archivo `definiciones.pas` contiene constantes, tipos y procedimientos auxiliares definidos por los docentes. El programa principal para realizar pruebas es provisto por los docentes en el archivo `principal.pas`. No se debe modificar

ninguno de estos archivos, dado que no formarán parte de la entrega. Para usar este programa se debe leer y seguir las instrucciones provistas en la sección Cómo ejecutar los casos de prueba de la Segunda Tarea.

También se provee el archivo `tarea2.pas`, que contiene los cabezales de los subprogramas a implementar. No se deben modificar los cabezales, sí se pueden definir variables locales, subprogramas, etc.

Evaluación

La entrega del laboratorio es **obligatoria** y el hecho de no entregar en fecha un archivo `tarea2.pas` tal que el programa principal compile implica la **pérdida del curso**.

Se aplicará una serie de casos de prueba a la entrega y se publicarán sus resultados, a manera de devolución. También se realizará una inspección automática (simple) de los códigos, analizando si se utilizan las estructuras correctas y si se aplican buenas prácticas de programación.

Para la aprobación del laboratorio, es **obligatorio** que los siguientes casos de prueba ejecuten correctamente (es decir, que su resultado sea igual al resultado esperado definido por los docentes):

- usuarios_02
- usuarios_03
- cifrado_01
- descifrado_01
- agregar_listar_05
- agregar_recuperar_03

De todas maneras, se **sugiere muy fuertemente** intentar llegar a los mejores resultados posibles.

Adicionalmente, en el parcial se incluirá un ejercicio que estará **muy inspirado** en la lógica de alguno de los subprogramas que tienen que resolver en esta tarea.

Introducción

Un **gestor de contraseñas** es una aplicación que permite a los usuarios almacenar todas sus credenciales (usuarios, contraseñas, sitios web a los que corresponden, etc.) en una base de datos cifrada mediante una contraseña "maestra". De este modo, el usuario puede gestionar todas sus cuentas desde una misma herramienta, memorizando únicamente una clave maestra.

La contraseña maestra es definida por cada usuario y permite que solo el dueño de la contraseña pueda acceder a los datos cifrados. Para cifrar los datos se pueden utilizar diferentes algoritmos de cifrado.

El **cifrado Vigenère** es un método de cifrado de texto alfabético donde cada letra del texto plano se codifica con un cifrado **César** diferente, cuyo incremento está determinado por la letra correspondiente de otro texto, la clave.

Por ejemplo, si el texto plano es **ataque esta noche** y la clave es **clave**, entonces

- la primera letra *a* del texto plano se desplaza *2* posiciones en el alfabeto (porque la letra *c* de la clave es la letra en la posición *2* del alfabeto, contando desde cero), lo que produce *c*;
- la segunda letra *t* se desplaza *11* posiciones (porque la letra *l* de la clave es la letra en la posición *11* del alfabeto, contando desde cero), lo que produce la letra *e*, con circularidad;
- la tercera letra *a* no se desplaza.
- en caso de que la clave sea mas corta que el texto, se vuelve a repetir la clave y así sucesivamente hasta que se termine el texto a cifrar.

Si el destinatario del mensaje conoce la clave, puede recuperar el texto original invirtiendo este proceso. El cifrado de Vigenère es, por lo tanto, un caso especial de sustitución polialfabética.

Descripción del Gestor de Contraseñas

El objetivo de la Tarea 2 es trabajar en la implementación de un gestor de contraseñas que permita a diferentes usuarios almacenar un conjunto de servicios y sus contraseñas. Para cada usuario se tendrá el nombre de usuario, una contraseña *maestra* y una lista de servicios. A su vez cada servicio tendrá el nombre del servicio y la contraseña cifrada con la contraseña maestra del usuario.

Para el desarrollo del gestor de contraseñas, vamos a tener en cuenta, por un lado, los algoritmos de cifrado que permitirán almacenar en forma "segura" las contraseñas de los servicios. Por otro lado, todo acceso al gestor de claves debe de ser autenticado, es decir, antes de realizar cualquier acceso o modificación del gestor de contraseñas se debe de verificar que el usuario y su contraseña maestra son válidas.

Algoritmo de cifrado

Como ya se ha visto en la Tarea 1, en un cifrado César, cada letra del alfabeto se desplaza a lo largo de una cierta cantidad de lugares. Por ejemplo, en un cifrado César de desplazamiento 3, *a* se convertiría en *d*, *B* se convertiría en *E*, y así sucesivamente. El cifrado Vigenère tiene varios cifrados César en secuencia con diferentes valores de desplazamiento.

Por ejemplo, supongamos que el texto sin formato que se va a cifrar es

ataque esta noche

La persona que envía el mensaje elige una palabra clave y la repite hasta que coincida con la longitud del texto sin formato, por ejemplo, la palabra clave "clave":

claveclaveclavecl

Para los caracteres sucesivos del mensaje, se tomarán letras sucesivas de la cadena de claves y cada carácter del mensaje se cifrará utilizando la letra de la clave correspondiente. Siguiendo este algoritmo, el resultado es:

cealyg enxc njgjp

Mecanismo de autenticación

La autenticación es un proceso crucial, que consiste en que quien está accediendo al gestor de contraseñas es quien dice ser. Esto se divide en 2 grandes funcionalidades: + en primer lugar, se debe validar la clave maestra del usuario, en esta tarea dicha funcionalidad será provista por los docentes + en segundo lugar, hay que asegurarse que todas las funciones que permiten modificar/leer el gestor de contraseñas, antes de realizar la acción deben invocar a la funcionalidad de autenticación.

Funcionalidades

Se pretende que el gestor de contraseñas brinde las siguientes funcionalidades:

- **Agregar usuario**, permite agregar un nuevo usuario al gestor, definiendo cuál es su contraseña maestra.
- **Agregar servicio**, dado un usuario, permita agregar un nuevo servicio, definiendo el nombre y la contraseña. La contraseña debe de ser almacenada en forma cifrada utilizando el algoritmo de Vigenère y la contraseña maestra del usuario. Antes de agregar el servicio, se debe de autenticar al usuario.
- **Recuperar la contraseña de un servicio**, dado un usuario y el nombre de un servicio, se deberá devolver la contraseña en texto claro (descifrada) para dicho servicio. Se debe de autenticar al usuario antes de devolver la información.

- **Listar servicios**, dado un usuario, imprimir la lista de servicios que hay registrados en el gestor para el usuario.

Adicionalmente y con el fin de que el usuario pueda probar los mecanismos de cifrados, se brindarán las siguientes funcionalidades:

- **Cifrar contraseña (con Vigenère)**, que dado un Texto y una Clave, imprime en pantalla el resultado de aplicar el cifrado de Vignére sobre el *Texto* utilizando la clave ingresada.
- **Descifrar contraseña (con Vigenère)**, en forma opuesta al cifrado, en este caso se recibe un texto cifrado y una clave y el sistema imprime el texto claro (descifrado) luego de procesar el algoritmo de Vigenère sobre el texto cifrado con la clave ingresada.

Constantes y Tipos de Datos

Las constantes y tipos de datos son provistos por los docentes en el archivo *definiciones.pas* y **no deben ser modificados**.

Se definen las siguientes constantes, todas de valor entero mayor que cero:

```
MAX_LARGO_TEXTO = ...; { tamaño máximo de los textos }
MAX_LARGO_CLAVE = ...; { largo máximo de una clave }
MAX_USUARIOS = ...;    { cantidad máxima de usuarios en el
                         gestor }
```

Y los siguientes tipos de datos:

```
{ Una cadena de Texto que puede ser usada para nombres,
  contraseñas, etc.}
Texto = record
    tex : array[1..MAX_LARGO_TEXTO] of char;
    tope : 0..MAX_LARGO_TEXTO
    end;
```

```
{ Tipo de cadena de caracteres exclusivamente para
el uso de claves de cifrado }
Clave = record
    cla : array[1..MAX_LARGO_CLAVE] of char;
    tope : 0..MAX_LARGO_CLAVE
    end;
```

```
{ Representa una lista de servicios, donde en cada nodo se
  almacena un servicio, junto a su contraseña cifrada}
TServicios = ^TServicio;
TServicio = record
  nombreServicio: Texto;
  contraServCifrada: Texto; {Contraseña cifrada}
```

```
  sig : TServicios
end;
```

```
{ Representa un usuario del gestor de contraseñas }
TUsuario = record
  usuario: Texto;
  serviciosUsuario: TServicios
end;
```

```
{ Estructura principal, donde se almacenan los usuarios y sus
  servicios }
TGestorContrasenia = record
      usuarios: array [1..MAX_USUARIOS] of TUsuario;
      tope:0..MAX_USUARIOS
end;
```

```
{ Estructura auxiliar, para devolver un servicio }
TMaybeServ = record case existe:boolean of
                  true : (coserv : Texto);
                  false : ()
          end;
```

```
{ Estructuras auxiliares para devolver la contraseña
  en texto claro de un servicio }
{ El enumerado TResp tiene los valores:
  serv: para cuando el servicio existe
  noserv: para cuando el servicio no existe
  nocontra: cuando falla la autenticación del usuario }
TResp = (serv,noserv,nocontra);
TRes = record case resp:TResp of
         serv : (cserv:Texto);
         noserv : ();
         nocontra : ();
      end;
```

```
{ Las siguientes son estructuras auxiliares usadas para la
  autenticación de los usuairos }
{ Almacena la información de autenticación de un usuario
  del sistema: Nombre, clave de Ceasar para cifrar la
  master key y la master key cifrada }
TUsuarioClave = record
                     usuario : Texto;
                     desp : 0..25;
                     masterCifrada : Clave;
                  end;
```

```
{ Estructura donde se almacena la información de
  autenticación de los usuarios del gestor}
TAutenticacion = record
      usuarios: array [1..MAX_USUARIOS] of TUsuarioClave;
      tope:0..MAX_USUARIOS
end;
```

```
{ Respuesta al autenticar, en caso de autenticación Ok, se
  devuelve la clave utilizada para cifrar las claves de
  los servicios }
TRespAutenticacion = record case autenticacionOK : boolean of
                              true : (master : Clave);
                              false : ();
                        end;
```

Subprogramas Auxiliares Provistos

Las subprogramas auxiliares son provistos por los docentes en el archivo *definiciones.pas* y **no deben ser modificados**. Pueden ser utilizados tanto para la implementación de los subprogramas solicitados como para realizar pruebas.

La función `sustituirLetra` y el procedimiento `cifradoCesarClave` son utilizados internamente para el proceso de autenticación.

```
function sustituirLetra(letra : char;
                        desplazamiento : integer) : char;
{ Esta función devuelve la letra correspondiente a aplicar el
  desplazamiento sobre la letra pasada como parámetro}
```

```
procedure cifradoCesarClave(textoOrig : Clave;
                            desp : integer;
                            var textoObj: Clave);
{ Dada una clave (`textoOrig`) y un desplazamiento (`desp`), devuelve
  en `textoObj` la clave cifrada utilizando el algoritmo César }
```

El procedimiento `agregarInfoAutenticacion` es necesario para implementar la funcionalidad de autenticación. Cuando se implemente la funcionalidad de `agregarUsuario` se deberá asegurar que se invoque correctamente esta función. Tiene como *precondición* que el usuario no existe y hay espacio en la infraestructura para almacenar su información. Las precondiciones son condiciones que se asume serán cumplidas al invocarse el subprograma, por lo que **no** deben chequearse.

```
procedure agregarInfoAutenticacion(us : Texto;
                                   cl : Clave;
                                   var authInfo : TAutenticacion);
```

```
{ pre : el usuario us no existe y hay lugar en authInfo }
{ Esta función recibe el nombre de un usuario y su clave maestra y
  lo almacena en la estructura auxiliar authInfo para poder
  autenticarlo más adelante }
```

Este procedimiento debe ser invocado en los lugares donde se deba autenticar a un usuario.

```
procedure autenticarUsuario (usuario : Texto;
                             clave : Clave;
                             authInfo : TAutenticacion;
                             var resp : TRespAutenticacion);
{ Dado un usuario y su clave maestra, devuelve en autenticacionOK
  true en caso de que el usuario exista y la contraseña sea válida.
  En ese caso además deuvleve en master la clave que debe utilizarse
  para cifrar y descifrar las contraseñas de los servicios }
```

Las funciones `igualTexto` e `igualClave` permiten comparar los tipos Texto y Clave.

```
function igualTexto(tx1,tx2 : Texto) : boolean;
{ Devuelve True si los dos textos ingresados son iguales,
  false en caso contrario }
```

```
function igualClave(cl1,cl2 : Clave) : boolean;
{ Devuelve True si las dos claves ingresadas son iguales,
  false en caso contrario }
```

Se proveen subprogramas de **escritura**, que permiten imprimir en la salida la información de algunos de los tipos definidos en la tarea. Estos son `desplegarTexto`, `desplegarClave`, `desplegarServicios`, `desplegarUsuarioClave` y `desplegarAuthInfo`. Estos subprogramas **no** deben usarse en el archivo `tarea2.pas` a entregar, pero pueden ser usados para realizar pruebas antes de la entrega.

Se proveen subprogramas de **lectura**, que permiten leer de la entrada la información de algunos de los tipos definidos en la tarea. Estos son `leerTexto` y `leerClave`. Estos subprogramas **no** deben usarse en el archivo `tarea2.pas` a entregar, pero pueden ser usados para realizar pruebas antes de la entrega.

Subprogramas Solicitados

Se deben implementar los siguientes subprogramas.

Procedimiento `cifradoVigenere`.

```
procedure cifradoVigenere (textoPlano : Texto;
                           cl : Clave;
                           var textoCifrado : Texto);
```

Aplica el cifrado Vigenère sobre el `textoPlano` utilizando la clave `cl` y lo devuelve en la variable `textoCifrado`.

Precondiciones:

1. el textoPlano tiene al menos un caracter.
2. la clave tiene al menos un caracter.

Procedimiento `descifradoVigenere`.

```
procedure descifradoVigenere  (textoCifrado : Texto;
                               cl : Clave;
                               var textoPlano : Texto);
```

Aplica el proceso opuesto, es decir, a partir de un `textoCifrado` se aplica el algoritmo de Vigenère para descifrarlo utilizando la clave `cl` y lo devuelve en la variable `textoPlano`.

Precondiciones:

1. el textoCifrado tiene al menos un caracter.
2. la clave tiene al menos un caracter.

Procedimiento `crearGestor`.

```
procedure crearGestor(var gc : TGestorContrasenia;
                      var authInfo : TAutenticacion);
```

Inicializa las principales estructuras del gestor de contraseñas, indicando que no hay usuarios registrados.

Procedimiento `agregarUsuario`.

```
procedure agregarUsuario( us : Texto;
                          cl : clave;
                          var gc : TGestorContrasenia;
                          var authInfo : TAutenticacion;
                          var full,existe : boolean);
```

Da de alta un usuario en el gestor de contraseñas. Recibe por parámetro el nombre del usuario (`us`), la clave maestra (`cl`), el gestor de contraseñas (`gc`) y la información de autenticación (`authInfo`). La variable `full` devuelve true en caso de que el gestor de contraseñas esté lleno y false en otro caso. La variable `existe` devuelve true en caso de que el usuario ya existe en el gestor de contraseñas, false en otro caso. En caso de que el usuario exista o el gestor de contraseñas este lleno, no se debe de dar de alta el usuario.

Precondiciones:

1. us tiene al menos un caracter.
2. cl tiene al menos un caracter.

Procedimiento `agregarServicioUsuario`.

```
procedure agregarServicioUsuario (us : Texto;
                                  master : Clave;
                                  authInfo : TAutenticacion;
                                  servn : Texto;
                                  co : Texto;
                                  var gc : TGestorContrasenia;
                                  var res : TRes);
```

Da de alta un nuevo servicio para el usuario `us`. Para poder modificar el gestor, es necesario autenticar al usuario utilizando su contraseña maestra (`master`). Luego de autenticado se agrega el servicio siempre y cuando ya no exista. Se recibe por parámetro el nombre del servicio `servn` y su contraseña `co` que debe de ser almacenada de forma cifrada.

En la variable `res` se devuelve el resultado de la operación, que puede ser: - si el usuario no existe o falla la autenticación, en `resp` se devuelve `nocontra` - si el servicio ya existe, no se crea y en `resp` se devuelve `noserv` - en caso de que todo funcione, en `resp` se devuelve `serv`

Procedimiento `contraseniaServicio`.

```
procedure contraseniaServicio (us : Texto;
                               master : Clave;
                               servn : Texto;
                               gc : TGestorContrasenia;
                               authInfo : TAutenticacion;
                               var res : TRes);
```

Dado un usuario, un servicio y el gestor de contraseñas, se autentica al usuario y en caso de que sea un usuario válido, se busca el servicio. En caso de que se

encuentre el servicio, se debe de descifrar la contraseña y se devuleve al usuario.

En la variable `res` se devuelve el resultado de la operación, que puede ser: - si el usuario no existe o falla la autenticación, en `resp` se devuelve `nocontra` - si el servicio no existe, en `resp` se devuelve `noserv` - en caso de que todo funcione, en `resp` se devuelve `serv` y en `cserv` la contraseña del servicio en texto claro

Procedimiento `serviciosUsuario`.

```
procedure serviciosUsuario(us : Texto;
                           master : Clave;
                           gc : TGestorContrasenia;
                           authInfo : TAutenticacion;
                           var servs : TServicios;
                           var existe : boolean);
```

Devuelve los servicios para un usuario. En primer lugar, se debe de autenticar al usuario, en caso de que sea un usuario válido, se debe de buscar y devolver la lista de servicios que tiene el usuario registrado en la variable `servs` en el orden en que fueron ingresados por el usuario. La variable `existe` devuelve true si el usuario existe y se autenticó correctamente, false en otro caso.

Se pide

Escribir en el archivo **tarea2.pas** todos los subprogramas solicitados. Los encabezados de los subprogramas **deben coincidir exactamente** con los que aparecen en esta letra. Si el estudiante realiza algún cambio se considerará que el subprograma no fue implementado. Si el estudiante lo desea, puede implementar subprogramas auxiliares adicionales (además de los subprogramas pedidos).

Para la corrección, las tareas se compilarán con una versión de Free Pascal igual o posterior a **3.0.4 para Linux**. La compilación y la ejecución se realizarán en línea de comandos. El comando de compilación se invocará de la siguiente manera:

```
fpc -Co -Cr -Miso -gl principal.pas
```

`principal.pas` será el programa principal entregado por el equipo docente.

NO se debe compilar con el IDE de Free Pascal.

No está permitido utilizar facilidades de Free Pascal que no forman parte del estándar y no se dan en el curso. Así por ejemplo, no se puede utilizar ninguna de las palabras siguientes: `uses`, `crlscr`, `gotoxy`, `crt`, `readkey`, `longint`, `string`, `break`, `exit`, etcétera.

En esta tarea, como en todos los problemas de este curso, se valorará, además de la lógica correcta, la utilización de un buen estilo de programación de acuerdo a los criterios impartidos en el curso. De esta manera, se hará énfasis en buenas prácticas de programación que lleven a un código legible, bien documentado y mantenible, tales como:

- indentación adecuada,
- utilización correcta y apropiada de las estructuras de control,
- código claro y legible,
- algoritmos razonablemente eficientes,
- utilización de comentarios que documenten y complementen el código,
- utilización de constantes simbólicas,
- nombres mnemotécnicos para variables, constantes, etcétera.

Para resolver la tarea se pueden utilizar todos los conocimientos vistos en el curso.