Laboratorio 2024 de Programación 1

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 1º de julio**. 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 3 de julio**.

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 del 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. No se exigirán mínimos con respecto a los casos de prueba ni a los resultados de la inspección de código, pero se **sugiere muy fuertemente** intentar llegar a los mejores resultados posibles.

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 procesador de texto es un programa informático diseñado para la creación, edición, formato y manipulación de documentos de texto. Proporciona herramientas que permiten a los usuarios, entre otras cosas, escribir y dar formato a textos. Los procesadores de texto ofrecen una amplia gama de funciones, como búsqueda de texto, conteo de caracteres y palabras, etc.

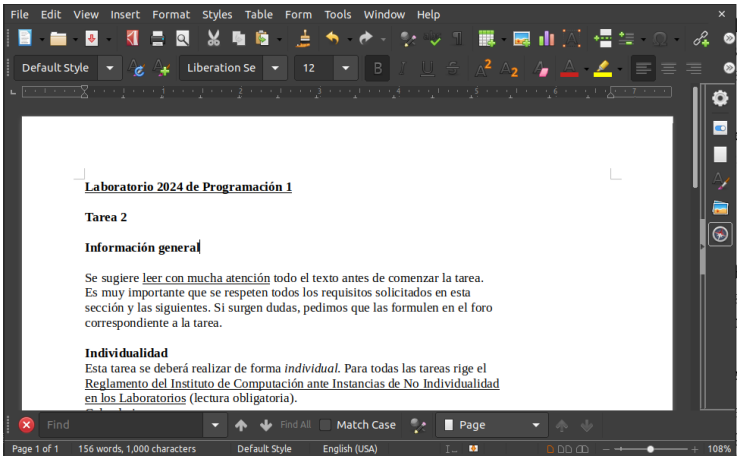

Figure 1: Ejemplo de Procesador de Texto

En este laboratorio implementaremos *Pascalígrafo*, una versión simplificada de un procesador de texto, que permite insertar textos compuestos por caracteres ASCII, decorar con negritas, itálicas y subrayados, realizar búsquedas y contar la cantidad de caracteres.

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:

```
MAXCOL = ...;       { cota de columnas de un archivo }
MAXCAD = ...;       { cota de cadena de caracteres }
```

Y los siguientes tipos de datos:

```
{ formato del texto }
TipoFormato = ( Neg, Ita, Sub );
Formato     = array [TipoFormato] of boolean;
```

```
{ un carácter en un texto incluye su formato }
Caracter = record
             car : char;
             fmt : Formato
           end;
```

```
{ arreglo con tope que representa a una línea  }
RangoColumna = 1..MAXCOL;
Linea    = record
             cars : array [RangoColumna] of Caracter;
             tope : 0..MAXCOL
           end;

PosibleLinea = record case esLinea :  boolean of
                 true  : (l: Linea);
                 false : ()
               end;
```

```
{ lista de líneas, que representa a un texto }
Texto    = ^NodoLinea;
NodoLinea    = record
                 info : Linea;
                 sig  : Texto
               end;
```

```
{ posición en el texto }
Posicion     = record
                    linea    : 1 .. maxint;
                    columna  : RangoColumna
                  end;

PosiblePosicion = record case esPosicion:  boolean of
                           true  : (p: Posicion);
                           false : ()
                         end;

PosibleColumna = record case esColumna:  boolean of
                           true  : (col: 1 .. MAXCOL);
                           false : ()
                        end;
```

```
{ cadena de caracteres }
Cadena       = record
                    cars : array [1..MAXCAD] of char;
                    tope : 0 .. MAXCAD
                  end;
```

Suprogramas 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 suprogramas solicitados como para realizar pruebas.

La función `ubicarLineaEnTexto` puede ser utilizada en la implementación de la tarea:

```
function ubicarLineaEnTexto ( txt: Texto; nln: integer ) : Texto;
{ Devuelve un puntero al texto en la línea numero `nln`, comenzando en 1.
  Si el texto no tiene una línea en la posición `nln`, devuelve `NIL`. }
```

Se proveen suprogramas de **escritura**, que permiten imprimir en la salida la información de algunos de los tipos definidos en la tarea. Estos son `mostrarCaracter`, `mostrarLinea`, `mostrarLineaCol`, `mostrarTexto`, `mostrarTipoFormato`, `mostrarFormato`, `mostrarFormatoTexto`, `mostrarPosiblePosicion`, `mostrarPosibleLinea` y `mostrarPosibleColumna`. 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 suprogramas de **lectura**, que permiten leer de la entrada la información de algunos de los tipos definidos en la tarea. Estos son `leerCadena`, `leerTipoFormatoy` `leerTexto`. 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. Las *precondiciones* son condiciones que se asume serán cumplidas al invocarse el subprograma, por lo que **no** deben chequearse.

Función `todosTienenFormatoEnLinea`.

```
function todosTienenFormatoEnLinea ( tfmt : TipoFormato
                                   ; ini, fin : RangoColumna
                                   ; ln : Linea ) : boolean;
```

Retorna `true` solo si todos los caracteres de `ln` entre las columnas `ini` y `fin`, incluídos los extremos, tienen el formato `tfmt`. En otro caso retorna false.

Precondiciones:

1. *1 <= ini <= ln.tope* (`ini` es una columna dentro de la línea)
2. *1 <= fin <= ln.tope* (`fin` es una columna dentro de la línea)

Procedimiento `aplicarFormatoLinea`.

```
procedure aplicarFormatoEnLinea ( tfmt : TipoFormato
                                ; ini, fin : RangoColumna
                                ; var ln : Linea );
```

Aplica el formato `tfmt` a los caracteres de `ln` entre las columnas `ini` y `fin`, incluídos los extremos. Si todos los caracteres ya tienen el tipo de formato `tfmt`, en lugar de aplicarlo lo quita.
Precondiciones:

1. *1 <= ini <= ln.tope* (`ini` es una columna dentro de la línea)
2. *1 <= fin <= ln.tope* (`fin` es una columna dentro de la línea)

Función `contarCaracteresEnTexto`.

```
function contarCaracteresEnTexto ( txt : Texto ) : integer;
```

Retorna la cantidad de caracteres que tiene el texto `txt`.

Procedimiento `buscarCadenaEnLineaDesde`.

```
procedure buscarCadenaEnLineaDesde ( c : Cadena; ln : Linea
                                   ; desde : RangoColumna
                                   ; var pc : PosibleColumna );
```

Busca la primera ocurrencia de la cadena `c` en la línea `ln` a partir de la columna `desde`. Si la encuentra, retorna en `pc` la columna en la que inicia.

Precondiciones:

1. *1 <= desde <= ln.tope* (`desde` es una columna dentro de la línea)

Procedimiento `buscarCadenaEnTextoDesde`.

```
procedure buscarCadenaEnTextoDesde ( c : Cadena; txt : Texto
                                   ; desde : Posicion
                                   ; var pp : PosiblePosicion );
```

Busca la primera ocurrencia de la cadena `c` en el texto `txt` a partir de la posición `desde`. Si la encuentra, retorna en `pp` la posición en la que inicia. La búsqueda no encuentra cadenas que ocupen más de una línea.

Precondiciones:

1. *1 <= desde.linea <= cantidad de líneas* (`desde` es una posición válida dentro del texto con respecto a la línea)
2. *1 <= desde.columna <= tope de línea en desde.linea* (`desde` es una posición válida dentro del texto con respecto a la columna)

Procedimiento `insertarCadenaEnLinea`.

```
procedure insertarCadenaEnLinea ( c : Cadena
                                ; columna : RangoColumna
                                ; var ln : linea
                                ; var pln : PosibleLinea );
```

Inserta la cadena `c` a partir de la `columna` de `ln`, y desplaza hacia la derecha los restantes caracteres de la línea. Los caracteres insertados toman el formato del carácter que ocupaba la posición `columna` en la línea. Si la columna es `ln.tope+1`, entonces queda sin formato. Si `c.tope + lin.tope` supera `MAXCOL`, los caracteres sobrantes se retornan (en orden) en la posible línea `pln`.

Precondiciones:

1. *1 <= columna <= ln.tope+1* (`columna` es una posición ocupada dentro de la línea o la posición libre inmediatamente posterior)
2. *c.tope + columna <= MAXCOL* (`c` entra completamente dentro de la línea, por lo que no hay que desplazar a ningún carácter de la cadena para `pp`)

Procedimiento `insertarLineaEnTexto`.

```
procedure insertarLineaEnTexto ( ln : Linea; nln : integer
                               ; var txt : Texto );
```

Inserta la línea `ln` en la posición `nln` del texto `txt`.

Precondiciones:

1. *1 < nln <= cantidad de líneas del texto + 1* (la línea siempre se va a insertar luego de una línea existente en el texto)

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.