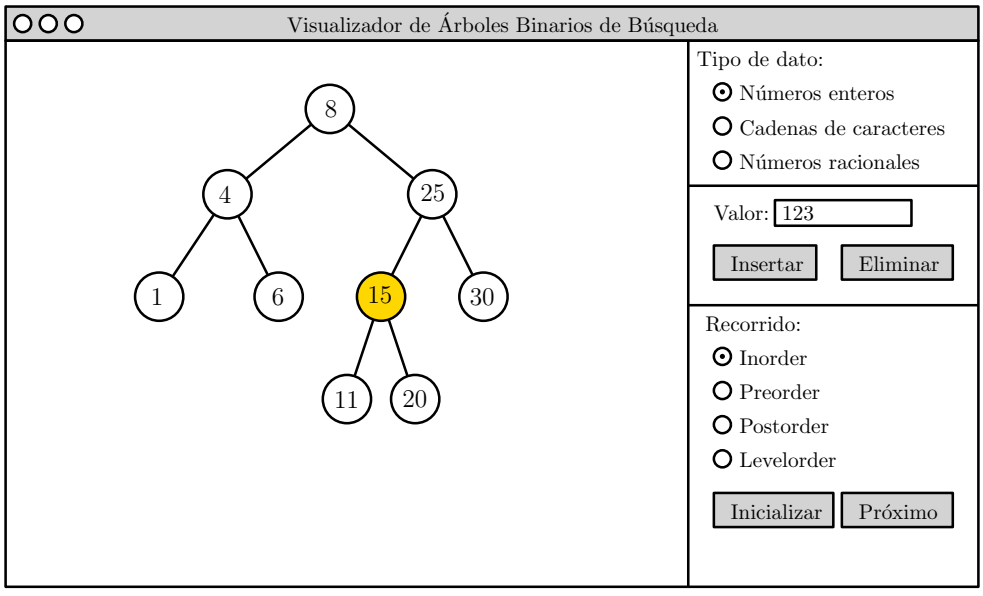
Paradigmas de Programación
Licenciatura en Ciencia de la Computación (USACH)

Trabajo de Laboratorio 3

(Fecha de entrega: 7 de julio de 2018) (Dos problemas, escoger uno!)

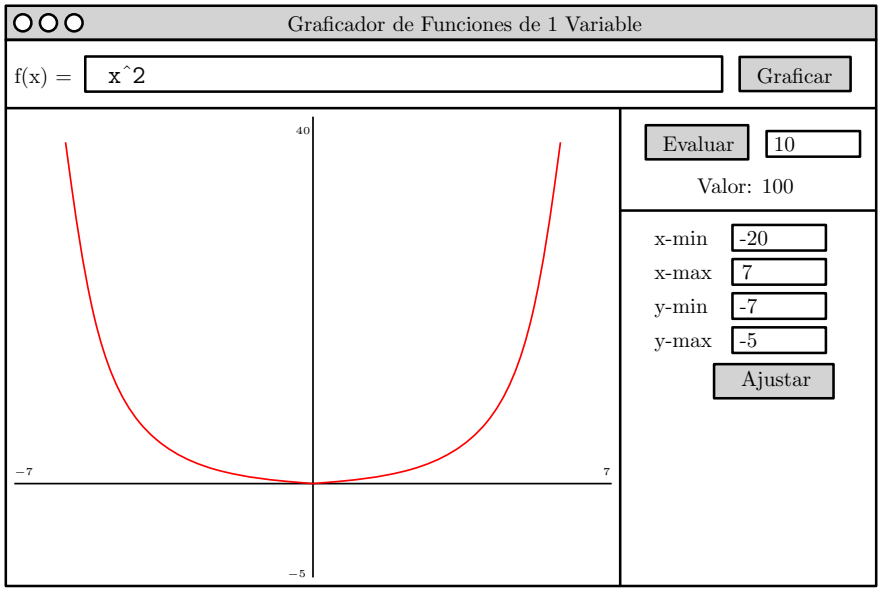
Problema 1. Visualizador de árboles binarios de búsqueda. Implementar una aplicación visual (ventana) en la que se visualicen árboles binarios de búsquedas, y se interactúen con los mismos a través de la inserción/eliminación de elementos, y los cuatro recorridos principales (inorder, preorder, postorder, y levelorder o por niveles). La ventana contará con la opción de elegir el tipo de dato a almacenar en el árbol (números enteros, cadenas de caracteres, o números racionales). Al abrirse la ventana por primera vez la opción elegida será la de números enteros, y cada vez que se cambie de opción se comenzará por un árbol vacío. Para insertar/eliminar elementos del árbol se usará una caja de texto para recoger el valor a insertar, y botones de "Insertar" y "Eliminar". Cuando se inserte/elimine un valor en el árbol, este debe ser redibujado en la ventana.

Para los recorridos, se contarán con cuatro opciones: inorder, preorder, postorder, y levelorder; siendo inorder la opción por defecto al abrirse la ventana. Existirá un botón "Inicializar" mediante el cual se inicializa el recorrido correspondiente escogido en la opción anterior. Con el botón "Próximo" el recorrido se moverá al siguiente (o primer) nodo del árbol, el cual deberá ser destacado en el dibujo del árbol. De llegar al final del recorrido no se destacará ningún nodo. Tener presente que si hay un recorrido activo no se puede eliminar el nodo actual del mismo, y que inserciones en el árbol, o eliminaciones de cualquier otro nodo no pueden afectar el recorrido que se esté efectuando.



Problema 2. Graficador de funciones de una variable. Implementar una aplicación visual (ventana) en la que se pueda entrar por teclado una función real en la variable `x`, y se permita: la graficación de la función; la evaluación de la función a través de ingresar un valor de la misma; y los ajustes de los ejes coordenados para una mejor visualización. La función que se entre por teclado debe ser cualquier expresión matemática (correctamente escrita), que involucre: la variable `x`, números enteros, números decimales, paréntesis, y operadores `+, -, *, /, **` (donde `**` es la exponenciación). Habrán bonificaciones adicionales para quienes agreguen funciones como: `exp(x), sin(x), cos(x), tan(x), ln(x), sqrt(x), abs(x)`, etc. Por ejemplo: `sin(2*x + 1) - (x+1)**2`. La idea es procesar el texto y representar la función mediante cierta estructura, de forma tal de no estar re-procesando el texto constantemente cada vez que se pretenda evaluar la función. Mediante el botón de "Ajustar" se debe cambiar el dibujo de la función para visualizar los ejes de acuerdo a los parámetros x-min, x-max, y-min, y y-max.

Es objetivo del trabajo programar una forma de procesar y evaluar la función, requiriendo un diseño de clases adecuado para representar la expresión de la función. No será considerado usar algo equivalente a la función `eval` de `Python`. Para procesar el texto de la expresión de la función, una técnica común es primeramente convertir la expresión de notación *infija* a notación *postfija* [`http://csis.pace.edu/~wolf/CS122/infix-postfix.htm`], ya que en esta última notación los paréntesis no se usan; para después procesar la notación postfija y constriuir la representación de la expresión de la función.



En la notación *infija* se escribe siempre el operador entre los dos operandos, por eso el *in*. En la notación *postfija* se escribe siempre el operador después de los dos operandos, por eso el *post*. Por ejemplo, supongamos (programando en `Python`) que tenemos la expresión `"2 * (30 + 7) + 5"`. Si la procesamos para extraer las cosas relevantes (tokens) tendríamos la lista `[2, "*", "(", 30, "+", 7, ")", "+", 5]`. Esta lista transformada a notación *postfija* daría la lista `[2, 30, 7, "+", "*", 5, "+"]`, desapareciendo los paréntesis. Evaluar esta lista es sencillo: Procesamos la lista de izquierda a derecha, manteniendo una lista S de operandos. Si el elemento actual de la lista es un operando, entonces se agrega al final de S. De lo contrario, si es un operador, se extraen los

dos últimos elementos de S, se les aplica el operador, y el resultado se pone al final de S como si fuera un operando. Al final, el único elemento que quede en la lista S será el valor de la expresión. El siguiente código `Python` realiza este algoritmo de evaluación:

```
def evaluate(postfix):
    S = []
    for e in postfix:
        if e in ["+", "-", "*", "/", "**"]:
            y = S.pop()
            x = S.pop()
            if e == "+":
                S.append(x + y)
            elif e == "-":
                S.append(x - y)
            elif e == "*":
                S.append(x * y)
            elif e == "/":
                S.append(x / y)
            else:
                S.append(x ** y)
        else:
            S.append(e)
    return S[0]
```

Reglas del Juego

(1) El trabajo se realizará en equipos de a dos. La nota será única, es decir, los miembros del equipo reciben todos la misma nota. Para la nota máxima del trabajo es necesario haber resuelto satisfactoriamente **uno** de los problemas, que cada integrante del equipo haya participado en la solución y la domine, y que además las solución sea original. **Como bonus, se pueden resolver ambos problemas!** y en cada uno se debe hacer un diseño de clases adecuado.

(2) La fecha de entrega es el día **domingo 7 de julio de 2019, hasta las 23:59 horas**. La fecha es **impostergable**. La entrega se hará por correo electrónico a la dirección del profesor. Los trabajos serán revisados por el profesor, con ayuda del ayudante, y se entregará la nota vía correo electrónico, junto con un reporte de los aspectos positivos y negativos del trabajo.

(3) En el programa tanto las estructuras de datos como los algoritmos tienen que ser eficientes, y esto suma puntos a la nota del trabajo.