Documento explicativo para Proyecto 1 de Matemáticas Discretas.

Link del repositorio de GitHub:
https://github.com/isabellitaxDioshija/ProyectoDiscretas.git.

Nombre: Isabella Garzón Salazar.

En este documento explicaré brevemente las funciones implementadas y los pasos que seguí para poder realizar el programa de función de valoración y probador de fórmulas.

- **Función str_to_list():** Con esta función se convierten las fórmulas en listas de caracteres, eliminando los espacios en blanco. Así, se puede trabajar directamente con los operadores y átomos sin preocuparnos por los espacios.
  Se usó un bucle que recorre la cadena y agrega los caracteres a una lista, ignorando los espacios. Esto permite que el evaluador no tenga que manejar espacios manualmente en cada operación, sino que de por sí ya los espacios fueron ignorados, de esta manera, sólo se conserva lo realmente importante para realizar las valoraciones, que son los átomos y los operadores.

- **Función V():** Como se necesitaba una función recursiva que evaluara expresiones lógicas con los operadores !, & y |, respetando la jerarquía de paréntesis (como lo que se explicó en la clase práctica con el paquete de Pringles).
  - Se implementó un condicional por si la expresión contiene un solo átomo, en este caso, sólo se devolvería su valor almacenado en el diccionario v, puesto que no estaríamos tratando con una fórmula sino con un átomo.
  - Se implementó un condicional por si la expresión comienza con "!" (símbolo de negación o not) en la cual se valora la fórmula y luego se niega la misma.
  - Y se implementó otro condicional para cuando la expresión está entre paréntesis. Cuando esto se cumple, se busca el operador principal (& o |) fuera de los paréntesis. Luego, se divide la expresión en dos partes y se aplican los operadores lógicos.

- **Función probarCombinaciones():** Esta función fue hecha pensando en una manera de probar todas las combinaciones posibles de valores de verdad para determinar si la fórmula era una tautología, una contradicción o una contingencia (lo que se pide en la parte b de la entrega).

  ¿Cómo se hizo?:

  Al principio intenté hacer todas las combinaciones de valores de verdad con un for, pero me di cuenta de que la función de recursión hacía el código más limpio y evitaba repetir muchas líneas. Básicamente, la función va probando todas las opciones (verdadero y falso) para cada átomo hasta que encuentra todas las combinaciones posibles. Para guardar estos valores se usó el diccionario v, que permite ir cambiándolos en cada intento sin perder el control de lo que se está probando.
- **Función analizarFormula():** Esta función se implementó ya que después de haber probado todas las combinaciones posibles de la fórmula, se necesitaba clasificar entre tautología, contingencia o contradicción.

  Se hizo de la siguiente manera: Si al probar todas las combinaciones siempre obteníamos True (primer condicional), significaba que la fórmula era una tautología (1). Si en todas daba False (segundo condicional), entonces era una contradicción (0). Pero si algunas combinaciones daban True y otras False (else), la fórmula era una contingencia (-1).
- **Función main():** En esta función se encuentra el menú para que el usuario pueda elegir la opción que guste: 1. Para probar el valor de verdad de una fórmula (Parte a de la entrega) y 2. Para probar si una fórmula es una tautología, contingencia o contradicción.

  Para permitir que el usuario elija qué acción realizar, se puso un input para leer su opción elegida. Como el programa también maneja múltiples entradas de datos, como los valores de verdad de los átomos y las fórmulas que se evalúan, usamos stdin.readline, que es más eficiente cuando se ingresan varias líneas seguidas.

  Luego, se organizó el flujo del programa con condicionales (if) de modo que, según la opción seleccionada, el código ejecute la función necesaria: Si el usuario elige la opción 1, se llama a la parte de valoración de fórmulas con valores fijos, y si elige la

opción 2, se ejecuta el probador de fórmulas para determinar si es una tautología, contradicción o contingencia. Se hizo con el fin de facilitarle al usuario la comprensión de lo que debe digitar de acuerdo a lo que quiera realizar.