REGLAS LÓGICAS DE INFERENCIA

Regla	Expresión
Declaración:	
Constante entera [INT]	i es una literal entera / ⊢i: int
Asignación de valores a una variable	O(x) = T / O⊢x: T
Arreglo	O[v]⊢v[i]: int / ⊢i: int
Argumento	O[param]⊢param: int / ⊢args: int
Símbolos especiales:	
Suma	⊢e1: int, ⊢e2: int / ⊢e1+e2: int
Resta	⊢e1: int, ⊢e2: int / ⊢e1-e2: int
División	⊢e1: int, ⊢e2: int / ⊢e1/e2: int
Multiplicación	⊢e1: int, ⊢e2: int / ⊢e1*e2: int
Mayor que	⊢e1: int, ⊢e2: int / ⊢e1<e2: int
Menor que	⊢e1: int, ⊢e2: int / ⊢e1>e2: int
Mayor igual que	⊢e1: int, ⊢e2: int / ⊢e1<=e2: int
Menor igual que	⊢e1: int, ⊢e2: int / ⊢e1>=e2: int
Diferente	⊢e1: int, ⊢e2: int / ⊢e1!=e2: int
Igual	⊢e1: int, ⊢e2: int / ⊢e1==e2: int
Función:	
Return [INT]	⊢función: int / ⊢return: int
Return [VOID]	⊢función: void / ⊢return: void

EXPLICACION DE LA ESTRUCTURA DE LA TABLA DE SÍMBOLOS

```
GlOBAL TABLE

Key             Type            Structure       Value
---             ----            ---------       ------
main            void            Function        void
sort            void            Function        ['Array', 'Const', 'Const']
minloc          int             Function        ['Array', 'Const', 'Const']
x               int             Array           10
hight           int             Const           None
low             int             Const           None
a               int             Array           None
```

Para la tabla me basé en el ejemplo del profesor, donde la **Key**, sirve para poder accesar a la información de cada fila además de que sirve para identificar el nombre de la variable.

El **Type** lo utilizo para poder validar los tipos de cada variable, y me sirve para checar si una variable está bien declarada (gracias a su tipo), si la función es del mismo tipo que su return, o si los argumentos de una llamada son iguales a los parámetros que recibe la función.

La **Structure** además de que nos ayuda a identificar el tipo de estructura del que se trata, ayuda para la declaración y llamada de funciones, ya que, si la función tiene como parámetros una structure de tipo 'Array', se iría a la parte de **Value** a buscar el arreglo que se crea, y en la posición en la que se manda el argumento, es la posición que se tiene en el array de value.

Por último, el **Value** sirve para ver que es lo que está almacenando la variable.

ESTRUCTURA DE DATOS

No ocupe el stack, sin embargo, utilicé una double linked list para poder recorrer por delante y por detrás los nodos (tablas) que se iban creando y fuera un tanto más fácil u tilizar esto.