Principio Básico de la programación

1. **Principio de la no Contradicción:** dos cosas lógicamente opuestas no pueden ser verdad al mismo tiempo, este concepto se conoce como el principio de no contradicción, un principio fundamental de la lógica que establece que una proposición no puede ser verdadera y falsa simultáneamente en la misma relación.

   Lógica formal: En lógica, esto significa que una proposición y su negación no pueden ser ambas verdaderas al mismo tiempo.

   En la lógica de programación aplico los siguientes pasos

   1. **Primero Analizo**
   2. **Después Resuelvo**
   3. **Al Final Programo**

   Analizar--------→ Resolver-----→Programar

   KISS (Keep It Simple, Stupid): Prioriza la simplicidad y claridad en el código.

   nota: entre más específico sea las instrucciones mejor será la ejecución de las tareas

   Conceptos Fundamentales:

   - **Algoritmo:** Una secuencia de pasos lógicos para resolver un problema.
   - **Variable:** Un espacio de memoria para almacenar datos que pueden cambiar durante la ejecución del programa.
   - **Función:** Un bloque de código que realiza una tarea específica y puede ser reutilizado.
   - **Tipos de datos:** Definen el tipo de información que puede almacenar una variable (números, texto, etc.).
   - **Estructuras de control:** Permiten controlar el flujo de ejecución del programa (condicionales, bucles).

2. **Análisis y resolución de problemas**

   Es un proceso sistemático para identificar, comprender y resolver problemas. Implica varias etapas, desde la definición del problema hasta la implementación de soluciones, con el objetivo de lograr resultados efectivos y eficientes.

a) Definición del problema:

- **Identificar el problema:** Es crucial entender claramente cuál es el problema central y sus causas subyacentes.
- **Recopilación de datos:** Recopilar información relevante para comprender el problema en profundidad. Esto puede incluir datos cuantitativos y cualitativos, así como información de diversas fuentes.

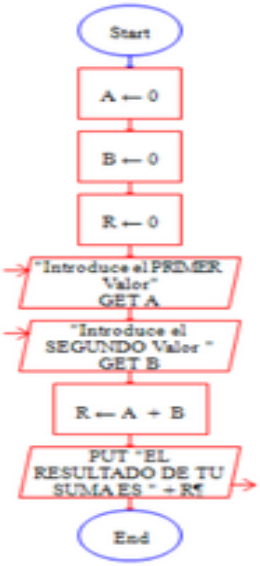
b) Análisis del problema:

- **Identificación de causas:** Analizar las causas raíz del problema, utilizando herramientas como el diagrama de Ishikawa (espina de pescado) o el análisis de 5 porqués.
- **Evaluación del contexto:** Comprender el entorno en el que ocurre el problema, incluyendo factores internos y externos que puedan estar influyendo.

Example: Realizar la operación A + B = R

ANALIZAR: se necesitan 3 espacios para la operación

RESOLVER: realizo la operación (+) A + B = R



Sentencias condicionales

los condicionales son estructuras que permiten tomar decisiones dentro de un programa, ejecutando diferentes bloques de código dependiendo de si se cumple o no una condición. Operadores relacionales

Operadores relacionales

Operador	Nombre	Ejemplo	Significado
<	Menor que	A < B	A es menor que B
>	Mayor que	A > B	A es mayor que B
==	Igual a	A == B	A es igual a B
!=	Diferente	A != B	A es diferente de B
<=	Menor que o igual a	A <= B	A es menor que o igual a B
>=	Mayor que o igual a	A >= B	A es mayor que o igual a B

Operadores Lógicos

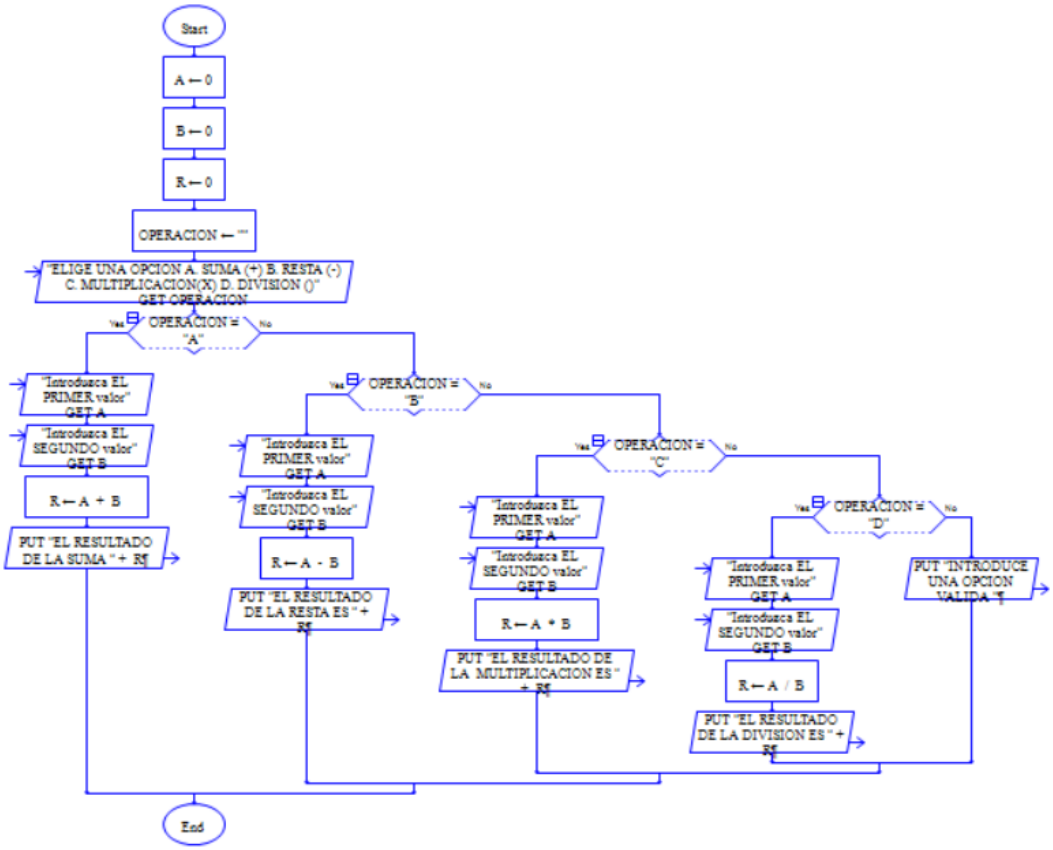
- && AND (el resultado es verdadero si ambas expresiones son verdaderas).
- || OR (el resultado es verdadero si alguna expresión es verdadera).
- ! NOT (el resultado invierte la condición de la expresión).

Ejemplos:

Problema: realizar una operación aritmética (+, -, *, /)

Análisis: necesito 4 espacios(variables); una variable para condicionar y 3 para los valores.

Resolver: preguntar qué operación realizar ("elige una opción a. suma (+) b. resta (-) c. multiplicación(x) d. división (/)"), después validar si se cumple alguna, si se cumple realizar la operación que corresponda, si no mensaje de alerta que la opción ingresada es invalida



3. Ciclos o Bucles

un bucle o ciclo: es una estructura que permite repetir un bloque de código varias veces, basándose en una condición. Los bucles son esenciales para automatizar tareas repetitivas y simplificar la programación.

un acumulador es una variable que se utiliza para almacenar la suma consecutiva de valores a lo largo de una serie de operaciones, generalmente dentro de un bucle.

Example:

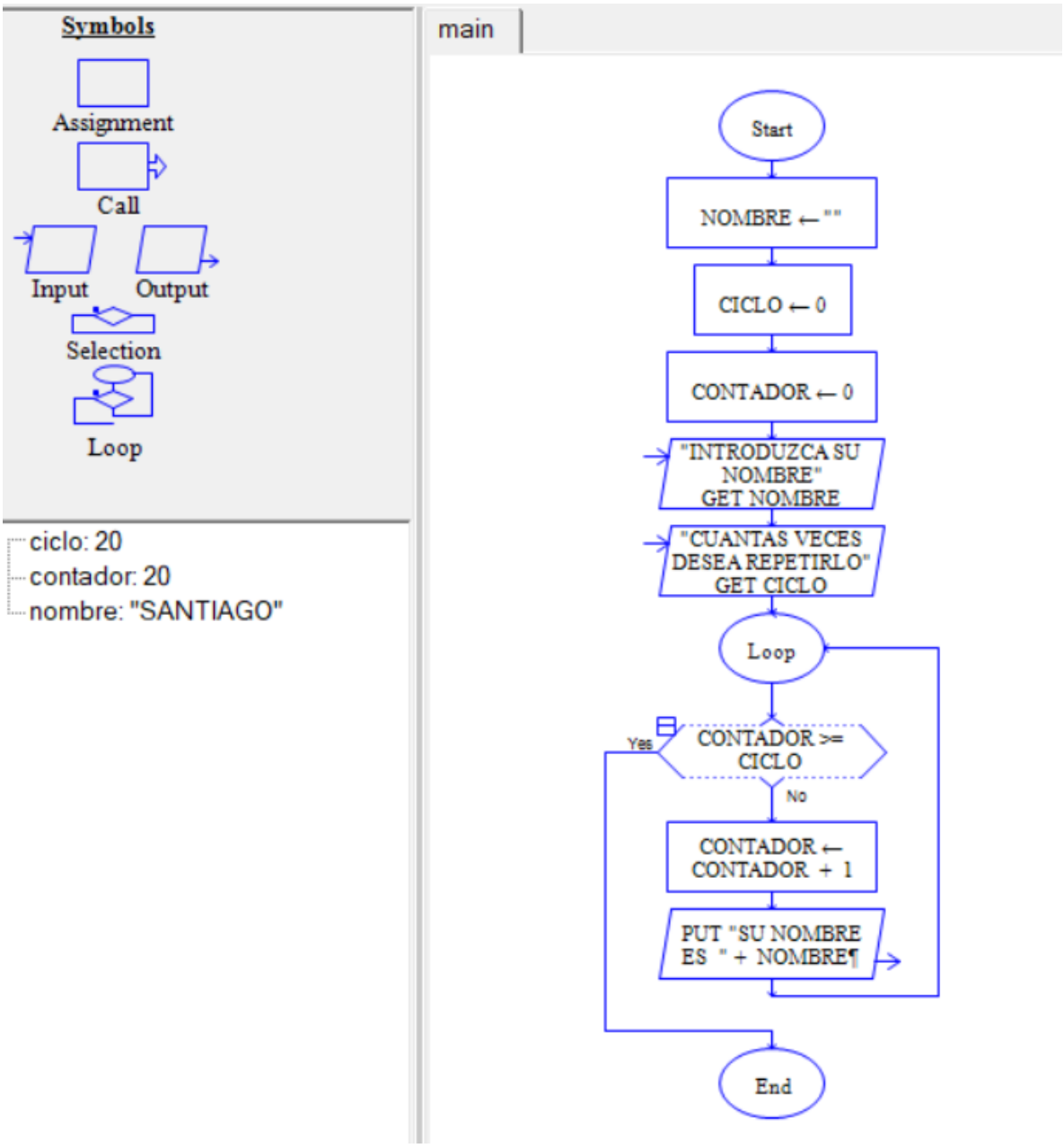
i= 0

repite este ciclo hasta que i >= 5

i= i + 1

sí i >= 5 entonces imprimir "MUCHAS GRACIAS"

DIAGRAMA DE FLUJO UTILIZANDO BUCLE

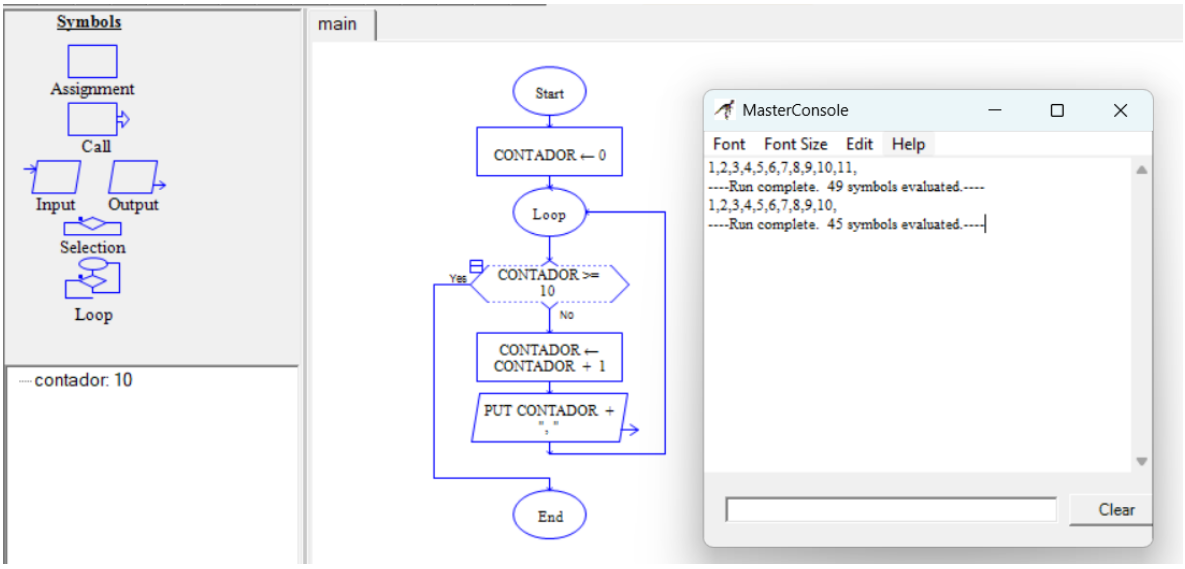


4. Desarrollo de la lógica

Una sucesión numérica es un conjunto de números ordenados que siguen un patrón o regla de formación.

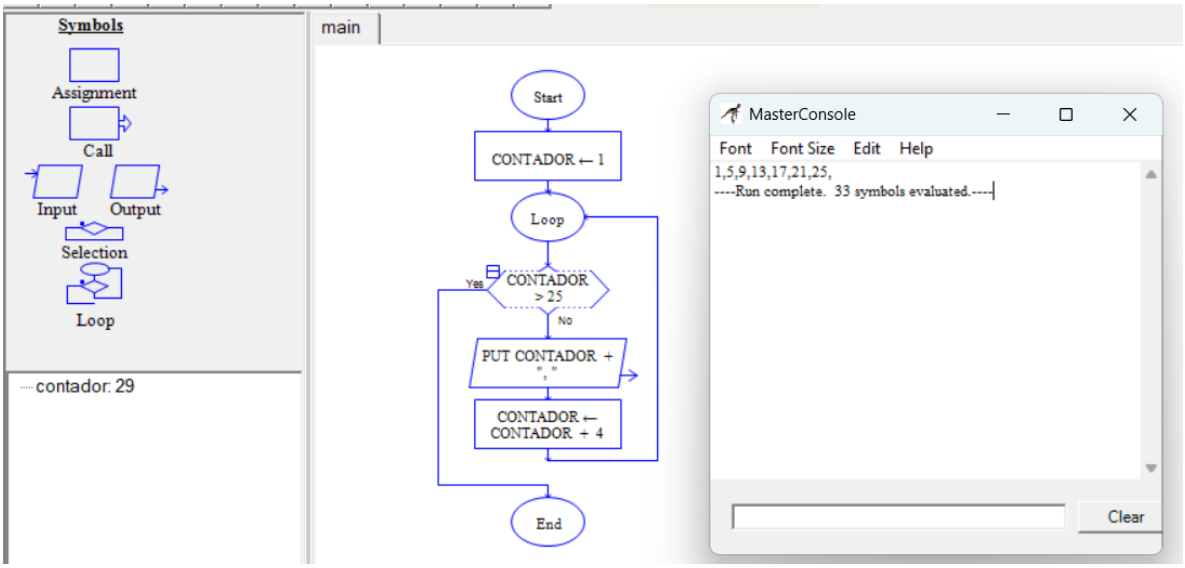
Ejercicios:

RESOLVER EL SIGUIENTE SERIE n veces 1, 2, 3, 4, 5, 6, 7, 8, 9, 10...



RESOLVER EL SIGUIENTE SERIE n veces 1, 5, 9, 13, 17, 21, 25...

los números van de 4 en cuatro



6. Sucesión Fibonacci

La sucesión de Fibonacci es una secuencia de números donde cada número es la suma de los dos anteriores, comenzando con 0 y 1

0, 1, 1, 2, 3, 5, 8, 13, 21, 34…

ANALISIS

RESULTADO=0, ANTERIOR = 1, POSTERIOR= 0

INICIAR HASTA QUE RESULADO > 34

1. IMPRIMIR RESULTADO + “,”
2. RESULTADO= ANTERIOR + POSTERIOR

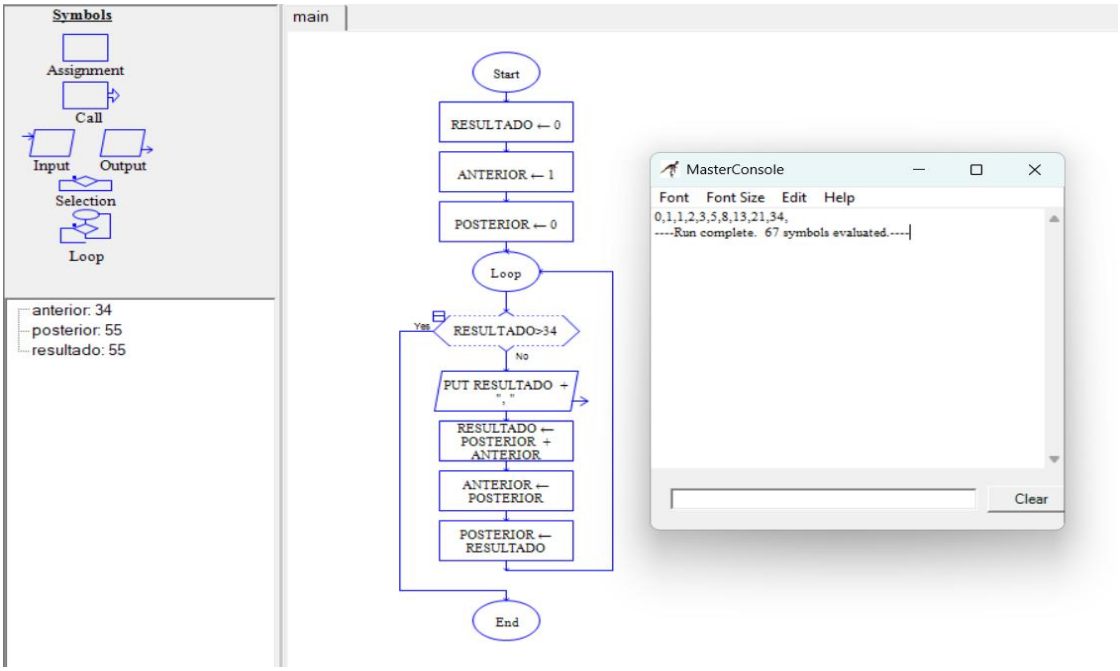
   0 = 1 + 0
3. ANTERIOR= POSTERIOR

   1 = 1
4. POSTERIOR = RESULTADO

   1 = 1

SI RESULTADO ES MAYOR 34 FIN

IMPRESO: 0, 1, 1,2,3,5,8,13, 21, 34



OTRO ANALISIS DE RESOLUCION

DOS VARIABLES PARA RESTAR EL ANTERIOR INICIALIZANDO

IMPRIMIENDO EL PRIMER VALOR DE R

R= 0, A= 1

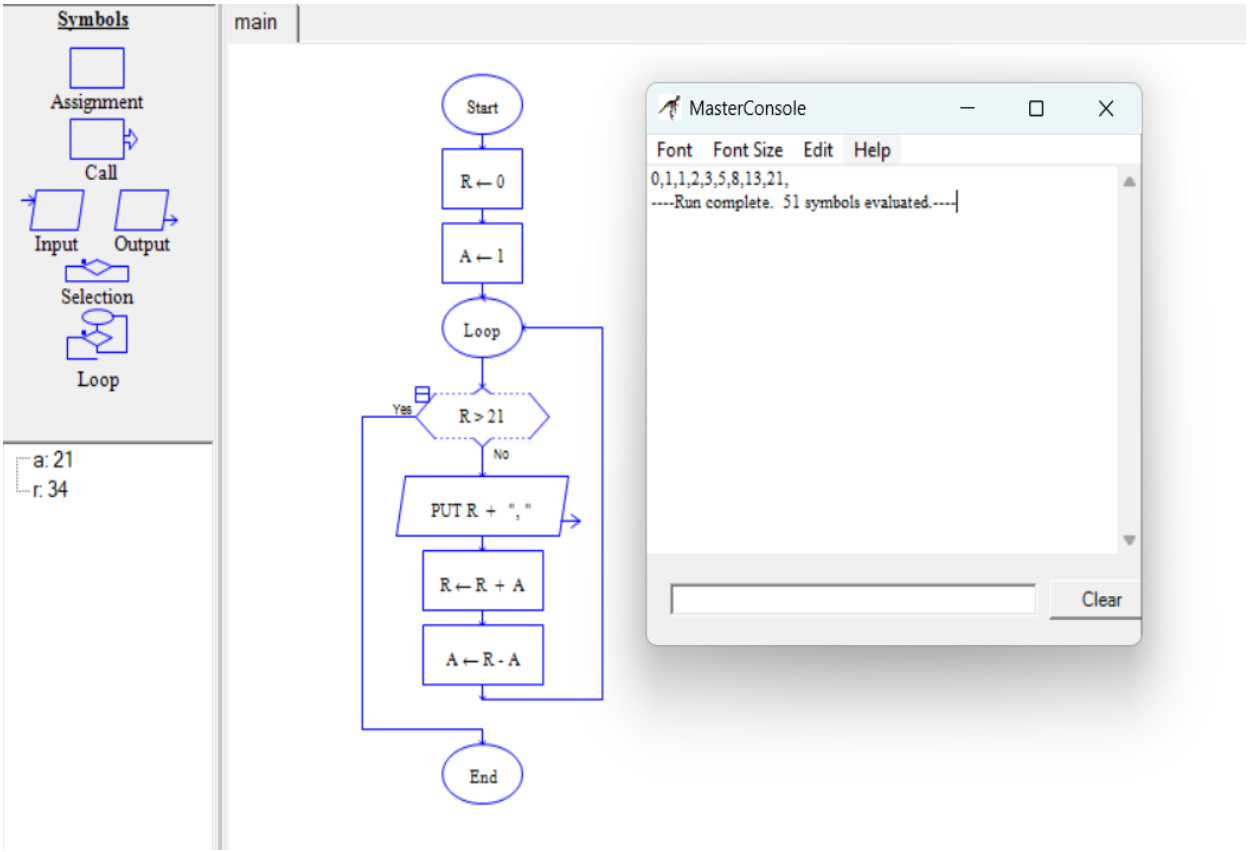
ENTONCES HACER MIENTRAS NO SEA R> 21

INICIAR

IMPRIMIR Contador + “,”

R = R + A

R = 0+ 1=1	**A=R - A**
1. R = 1 +0= 1	**1.** A = 1 - 1 = 0 --→ A = 0, R = 1
2. R = 1 +1 = 2	**2.** A=1 - 0 = 1 --→ A = 1, R = 1
3. R = 2+1 =3	**3.** A=2 - 1= 1--→ A= 1, R= 2
4. R = 3+2=5	**4.** A = 3- 1 =2 --→ A = 2, R=3
5. R =5+3= 8	**5.** A= 5 - 2 =3 ---→ A =3, R=5
6. R =8 +5 = 13	**6.** A= 8 - 3= 5 --→ A= 5, R= 8
7. R =13+8= 21	**7.** A= 13 - 5 = 8 -→ A = 5, R= 13



7. Programación Orientada a Objetos Principios básicos

Organiza el diseño de software alrededor de datos (objetos) en lugar de funciones y lógica. Se centra en la manipulación de objetos que representan entidades del mundo real, y en las interacciones entre ellos, en lugar de ejecutar instrucciones secuenciales.

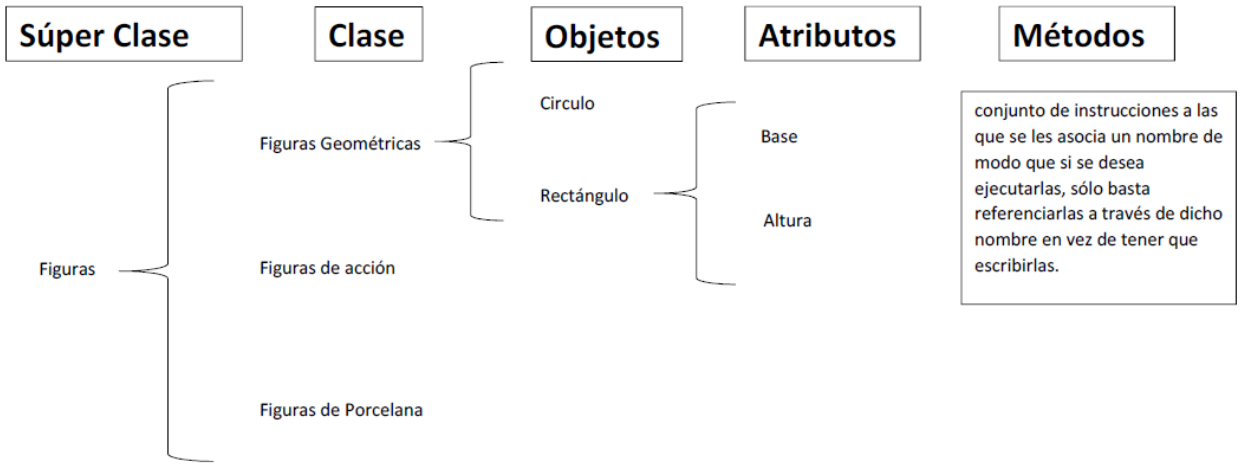
La programación de objetos permite el encapsulando datos (atributos) y comportamiento (métodos). Esto permite crear sistemas más modulares, reutilizables y fáciles de mantener.

Conceptos clave de la POO:

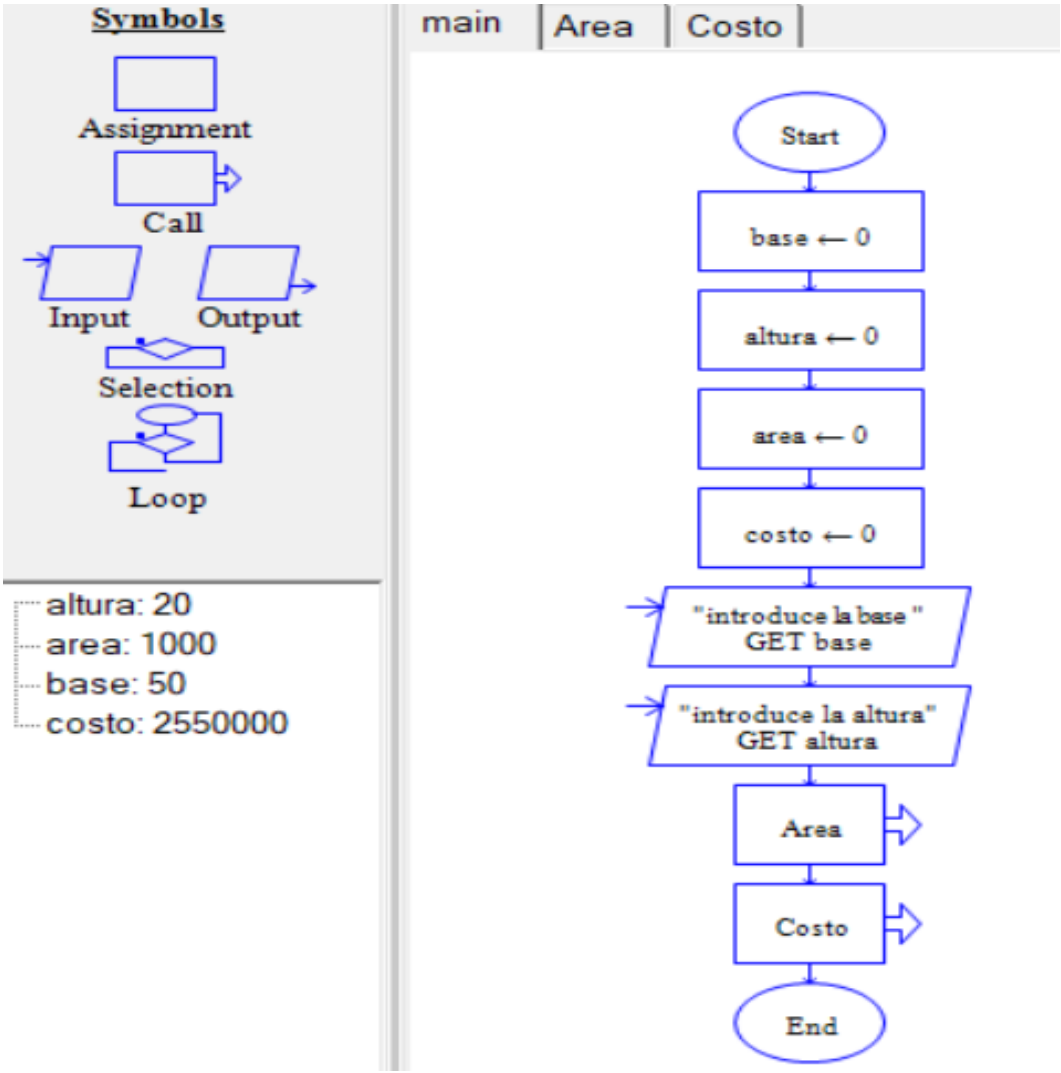
- **Clases:** Son plantillas o modelos que definen la estructura y el comportamiento de los objetos. Una clase es como el plano para crear objetos.
- **Objetos:** Son instancias de una clase. Cada objeto tiene su propia identidad, estado (datos) y comportamiento (métodos).
- **Atributo:** Una característica o propiedad de un objeto.
- **Método:** Una función asociada a una clase que define el comportamiento de sus objetos.
- **Encapsulamiento:** Oculta la implementación interna de un objeto, exponiendo solo una interfaz pública para interactuar con él. Esto ayuda a proteger los datos y evitar modificaciones no deseadas.
- **Herencia:** Permite que una clase herede atributos y métodos de otra clase (clase padre/base), creando una jerarquía de clases. Esto facilita la reutilización de código y la creación de relaciones entre objetos.
- **Polimorfismo:** Permite que objetos de diferentes clases respondan de manera diferente a la misma llamada de método. Esto brinda flexibilidad y dinamismo a los sistemas.

Diferencia entre FUNCION y METODO

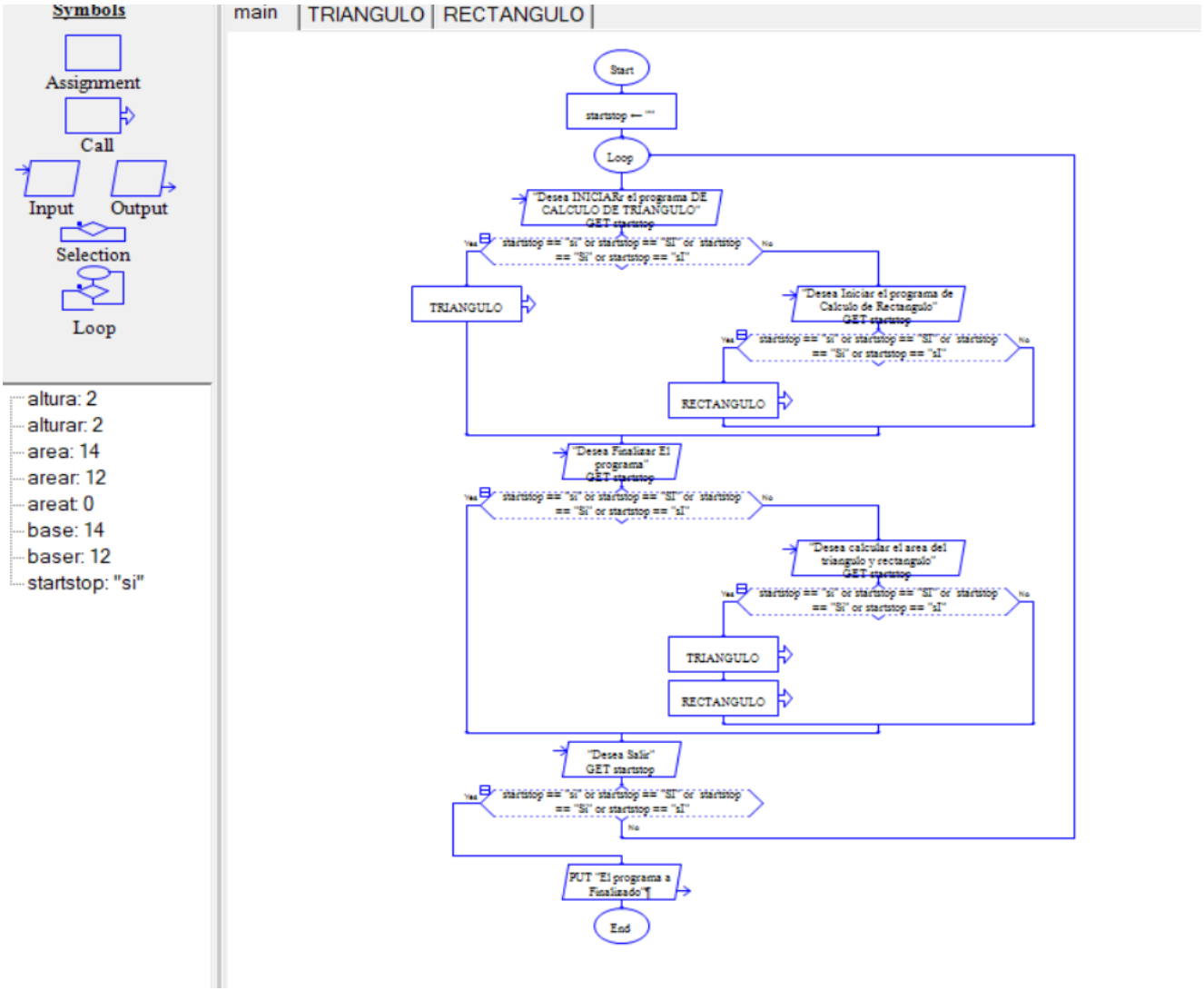
Una función es un bloque de código independiente que puede ser llamado directamente sin necesidad de un objeto, mientras que un método está siempre ligado a un objeto o clase, y puede acceder y modificar el estado de ese objeto.



Practica de llamado de objetos y métodos desde una super clase



llamado de objetos y métodos



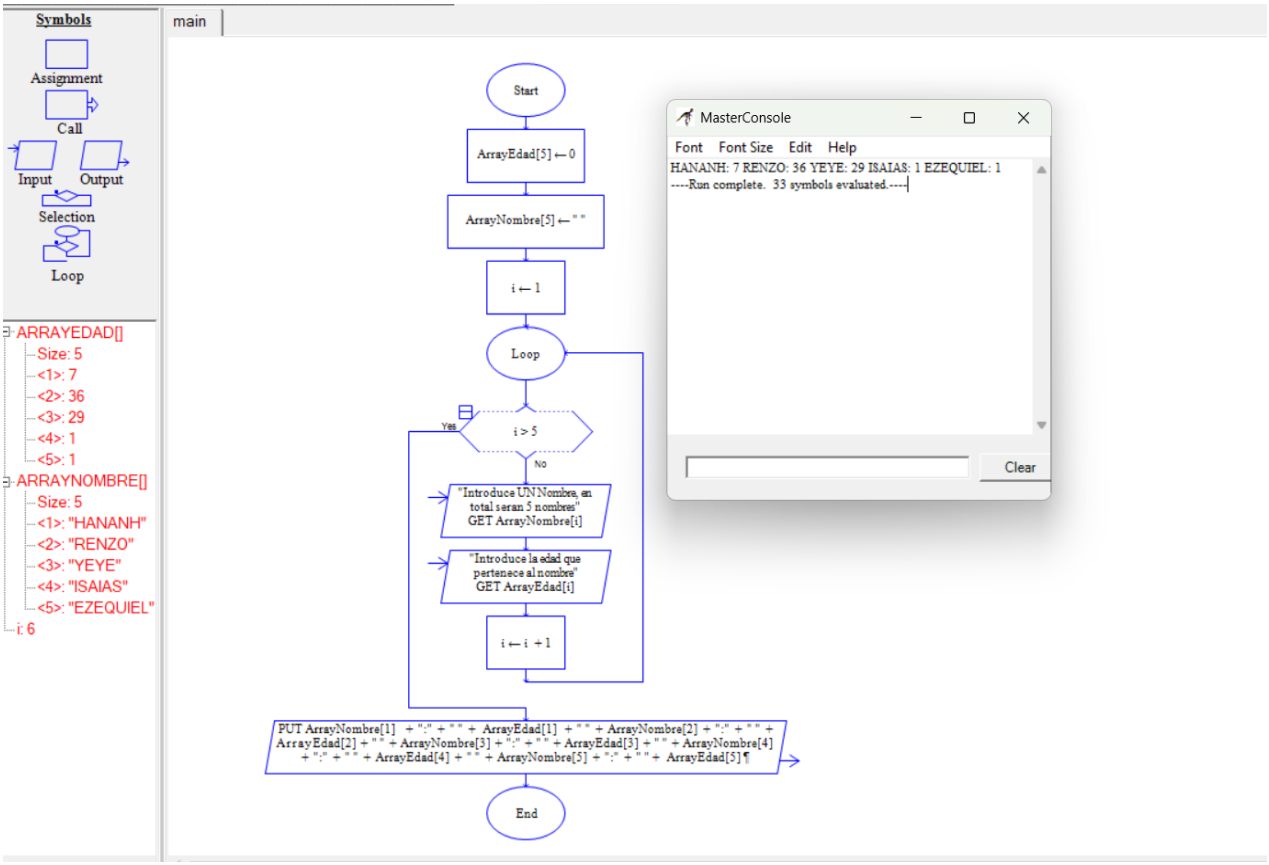
8. Arreglos unidimensionales & bidimensionales

En programación, un arreglo (también conocido como array o matriz) es una estructura de datos que almacena una colección ordenada de elementos del mismo tipo. Permite almacenar múltiples valores en una sola variable, accediendo a cada elemento a través de un índice numérico.

NOTA: básicamente son espacios en memoria en vez declarar mucha variable se puede realizar un arreglo y asignarle el espacio en memoria.

Estos espacios de datos se pueden ordenar en una sola dirección (unidimensional(fila)) o se pueden ordenar los datos bidimensionales (filas y columnas) y se pueden acceder a esos datos ubicando su posición en la matriz

Ejemplo: Arreglo ordenados los datos por nombre y edad



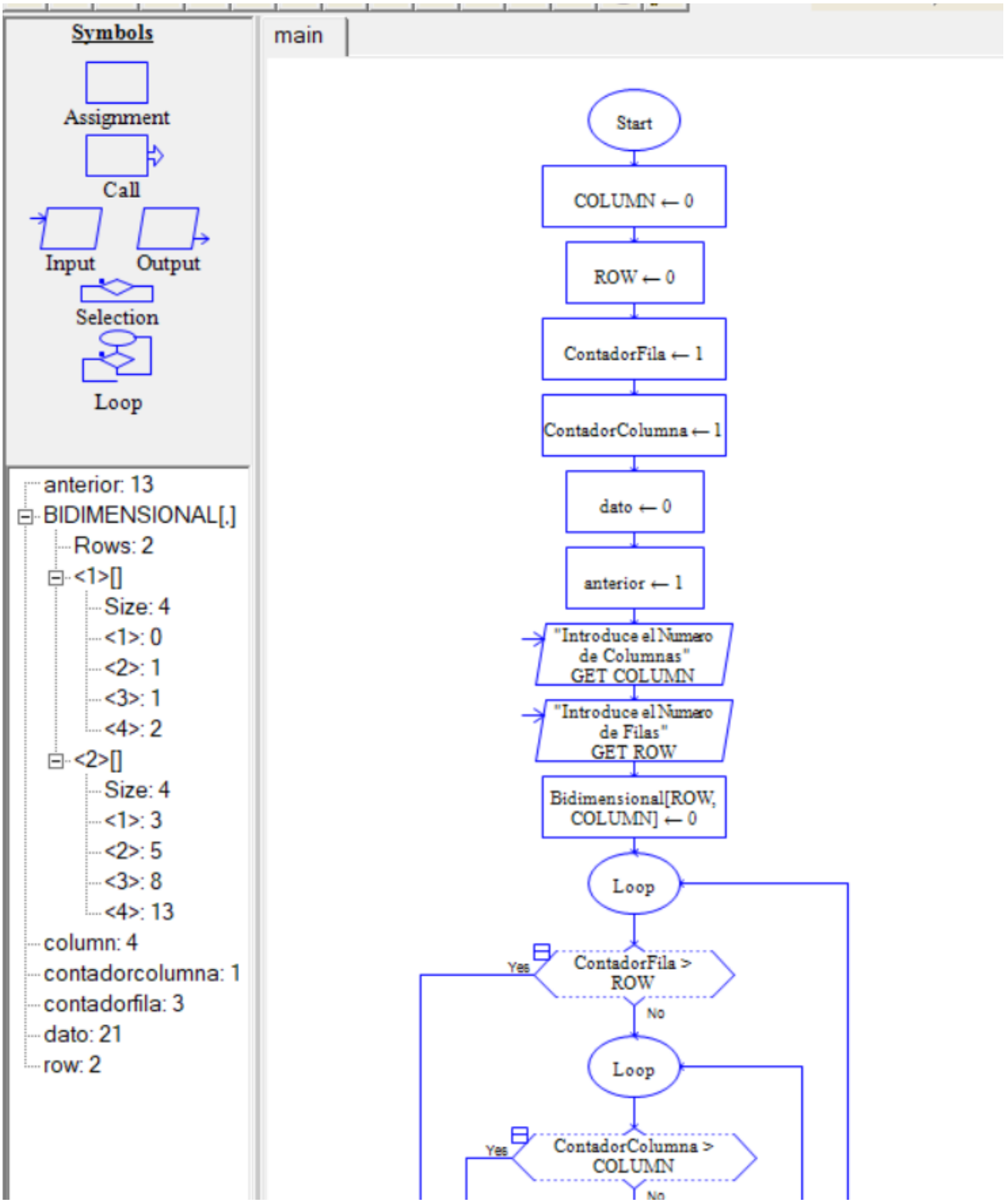
9. Arreglos dinámicos & estáticos

Un arreglo estático tiene un tamaño fijo definido en tiempo de compilación y no puede cambiar durante la ejecución del programa.

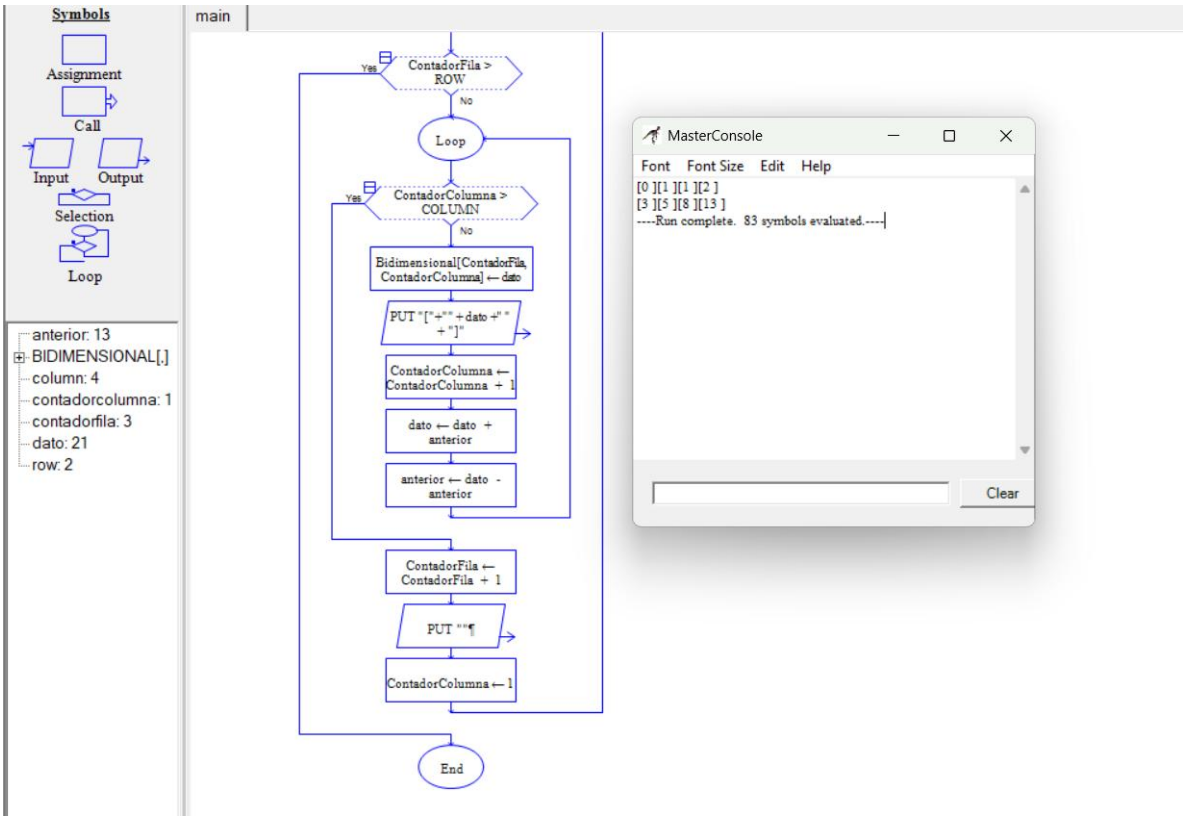
Un arreglo dinámico, por otro lado, permite que su tamaño se ajuste durante la ejecución, ya sea creciendo o disminuyendo según sea necesario.

10. Arreglos bidimensionales dinámicos

Ejercicio de un arreglo interactivo donde el usuario de termina la cantidad de filas y columnas, y guarda los datos en la posición del array de una serie de Fibonacci.



Análisis donde el primer bucle condiciona las filas y el segundo condiciona las columnas para que se vayan guardando los datos en la matriz



11. Estructura de datos (algoritmos de ordenamiento)

¿QUE ES UN ALGORITMO?

Un algoritmo es un conjunto finito de pasos, operaciones o instrucciones, definidos y ordenados, que permiten resolver un problema o realizar una tarea. En esencia, es una receta para lograr un objetivo específico, siguiendo una secuencia lógica.

Método de ordenamiento de burbuja

es un algoritmo simple para ordenar una lista de elementos. Funciona comparando elementos adyacentes e intercambiándolos si están en el orden incorrecto, repitiendo este proceso hasta que la lista esté ordenada.

Reglas

1. los apuntadores que rige nuestro arreglo deben de ir al final y regresar el mismo número de posiciones de nuestro array. es decir, si nuestro array tiene 5 posiciones nuestros apuntadores deben recorrerlo 5 veces

2. cada vez que se cumpla una condición asignada, se debe intercambiar los números. siempre que el número que tenga el **Apuntador1** sea menor que el **Apuntador2** se debe intercambiar **A > B**

3. Nunca se debe perder el número de vista durante el intercambio, se debe guardar en una variable auxiliar

12. Método de la burbuja con Raptor

Para la solución del método de ordenamiento de burbuja, se plantea la siguiente condición, si la posición uno es mayor a la posición dos, estos se deben intercambiar.

Entonces planteando la condición se aplican las 3 regla anteriores y definimos los siguiente:

Se declaran 4 variable y un arreglo definido

A = 1

B= 2

ARRAY [5] // donde defino el tamaño arreglo

saved= 0

vueltas = 0

1. primero creamos un bucle solicitando los datos al usuario

la condición del bucle es que termine cuando la posición **A > 5** ENTONCES

Pedimos los datos al usuario de cada posición en el arreglo **ARRAY[A]**

luego movemos esa posición e introducimos el siguiente dato a través de un contador **A=A+1**

imprimimos el resultado de cada posición

1. una vez definido el arreglo evaluamos la cantidad de veces que va recorrer ese arreglo a través de un bucle, en este caso está definido en 5 por lo tanto

Hacer los siguientes condicionales mientras el número de **Vueltas > 5**

si la posición B NO sea **B >5** ejecuta lo siguiente:

si la posición del arreglo **ARRAY[A] > ARRAY[B]** entonces

1. Guardamos esa posición

saved = ARRAY[A]

2. La posición que esta vacía ARRAY[A] le asignamos el valor de ARRAY[B]

ARRAY[A]= ARRAY[B]

3. La Posición de ARRAY[B] guarda el valor de saved

ARRAY[B] = saved

4. Genera desplazamiento a través de dos contadores para se muevan las posiciones de A Y B

A= A +1

B= B + 1

Repite el ciclo hasta que B NO sea **B >5**

SI es mayor entonces

cuenta una vuelta y reiniciamos las posiciones

vuelta = vuelta + 1

A=1

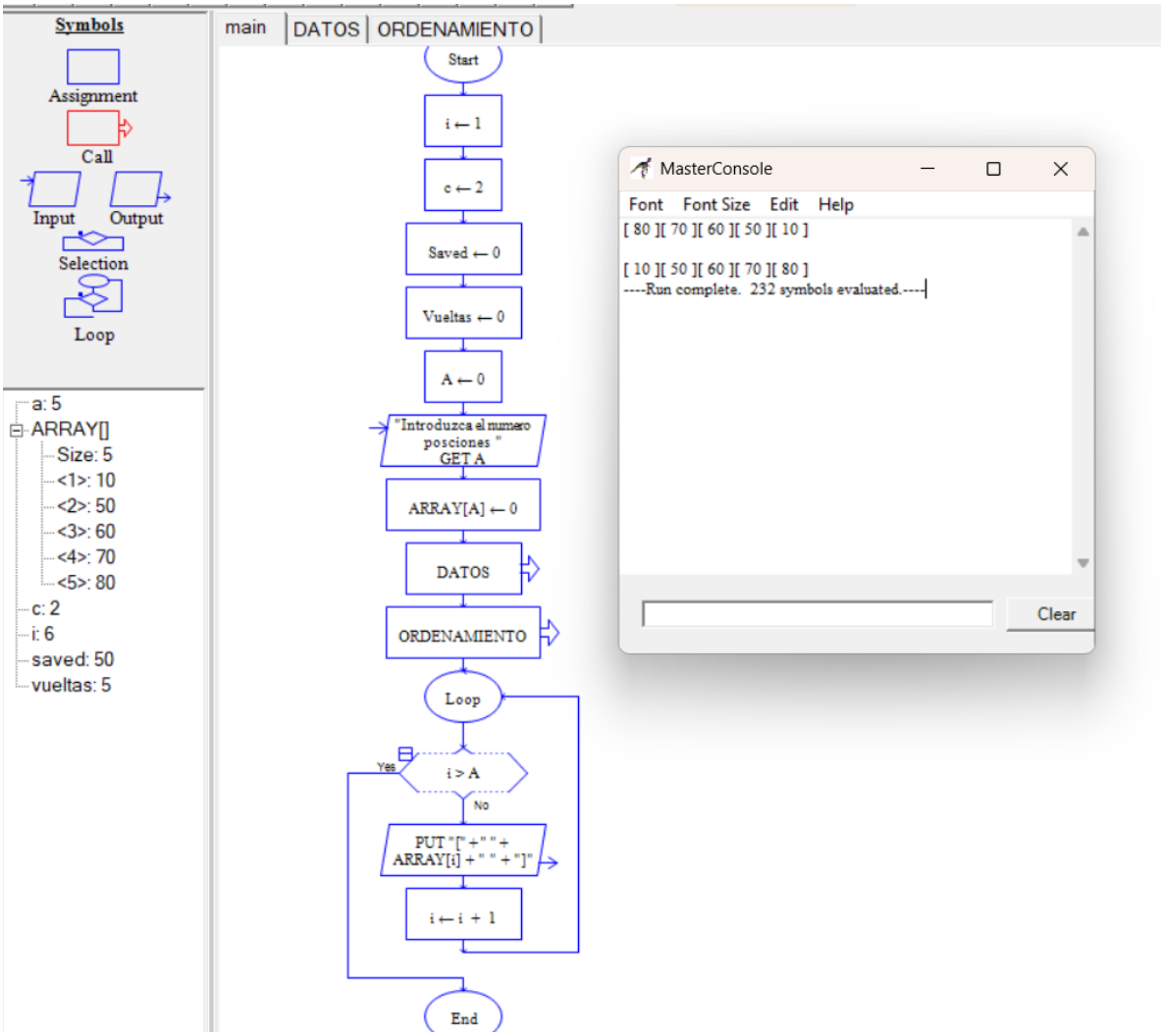
B = 2

si el número de **vueltas > 5** entonces

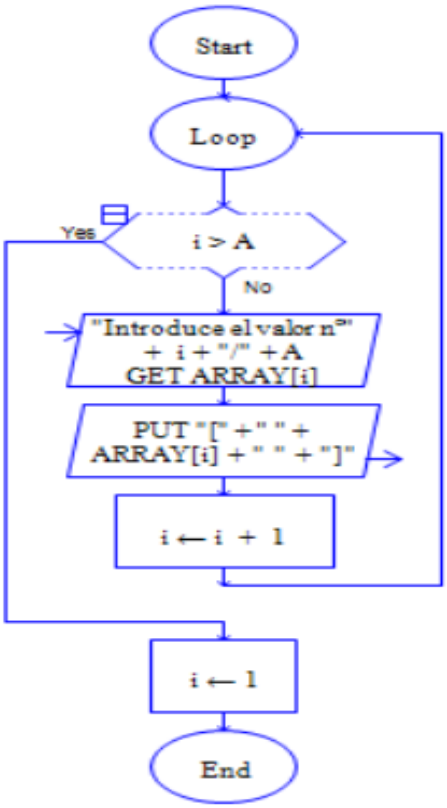
imprimimos en pantalla el arreglo y listo.

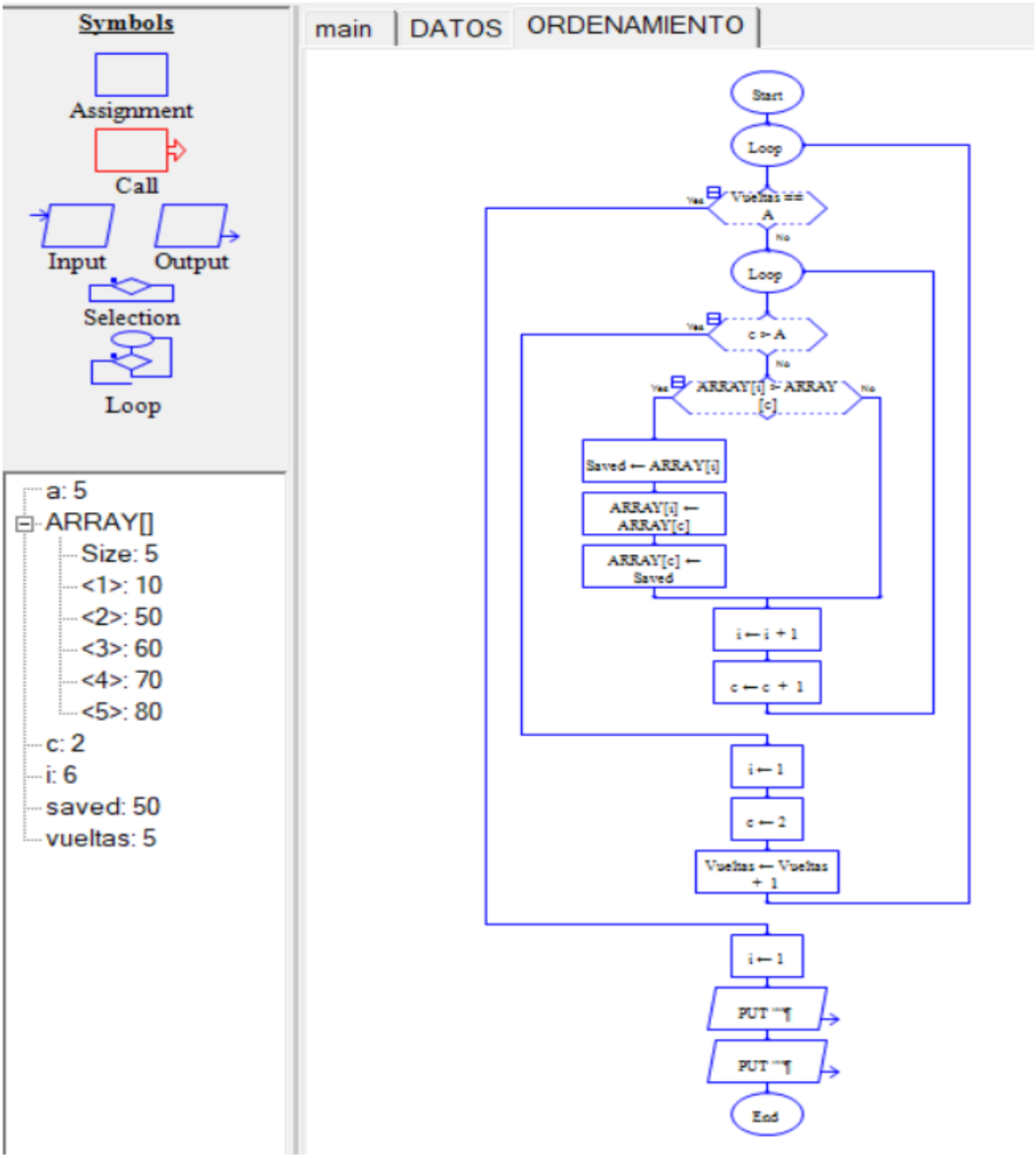
Ejercicio:

En este caso se solicita el número de posiciones y divide por métodos



main DATOS | ORDENAMIENTO |





13. Aplicación práctica del algoritmo Burbuja

Sistema de asignación de becas

Desarrollar un programa que identifique a los alumnos con el promedio necesario para ser beneficiarios de una beca con las siguientes características:

- Que solicite el nombre y calificación de cualquier cantidad de alumnos, y posteriormente que muestre un listado de manera descendente, es decir, desde el alumno con el mayor promedio hasta

el alumno con el promedio más bajo y que contenga lo siguiente: Nombre del alumno, su respectiva

- calificación y una leyenda que diga CON DERECHO A BECA o SIN BECA.

NOTA: Únicamente los alumnos con promedio mayor o igual a 8 podrán tener derecho a la beca.

Lista original en desorden:

1. Luis, calificación 8
2. Leslie, calificación 5
3. Gerardo, calificación 7
4. Natalia, calificación 10
5. Enrique, calificación 9
6. Fernando, calificación 6

ANALISIS:

NECESITAMOS 5 VARIABLES

A =1; REPRESENTARA LA POSCION 1 DEL ARRAY

B =2; REPRESENTARA LA POSICION DEL ARRAY

Vueltas =0 ;

SaveNombre= 0

SaveNotas= 0

Definimos dos arráis finitos

Nombre[6]

Notas[6]

Creamos un bucle para solicitar los datos al usuario NOMBRE Y NOTAS

INICIAR HASTA A > 6 ENTONCES

SE SOLITA LOS **DATOS DEL NOMBRE** **POSICION A**

SE SOLICITA LOS **DATOS DE LA NOTA** **POSCION A**

Y AUMENTAMOS LA POSICION A= A +1

IMPRIMIMOS LISTA DESORDENADA

UNA VEZ QUE TERMINE EL BUCLE YA TENEMOS NUESTRA LISTA Y APLICAREMOS OTRO BUCLE PARA RECORRERLA Y CONDICIONARLA

HACER MIENTRAS **Vueltas > 6** and **Nota[A]< Notas[B]** entonces

guardamos la posición:

SaveNota= Nota[A]

SaveNombre= Nombre[A]

y la posición:

Nota[A] = Nota[B]

Nombre[A]= Nombre[B]

Nota[B] = SaveNota

Nombre[B]= SaveNombre

Realizamos el desplazamiento de las posiciones

A= A+1

B= B +1

CUANDO B LLEGUE A LA ULTIMA POSICION QUE ES 6 CONTAR UNA VUELTA con otro BUCLE Y REINICIAR OTRA VEZ LA POSICON PARA QUE LA RECORRA DE NUEVO

A=1

B=2

Vuelta = Vuelta +1

hasta HACER el número de vueltas que componen las posiciones de nuestro array6 una vez tenido la lista

condicionamos si califica o no

PERO ANTES REINICIAMOS NUESTRA POSICION PARA RECORRERLA DESDE EL PRINCIPIO NUESTRO ARRAY

A=1

IF **Nota[A] > 8** ENTONCES CALIFICA

SINO ELSE

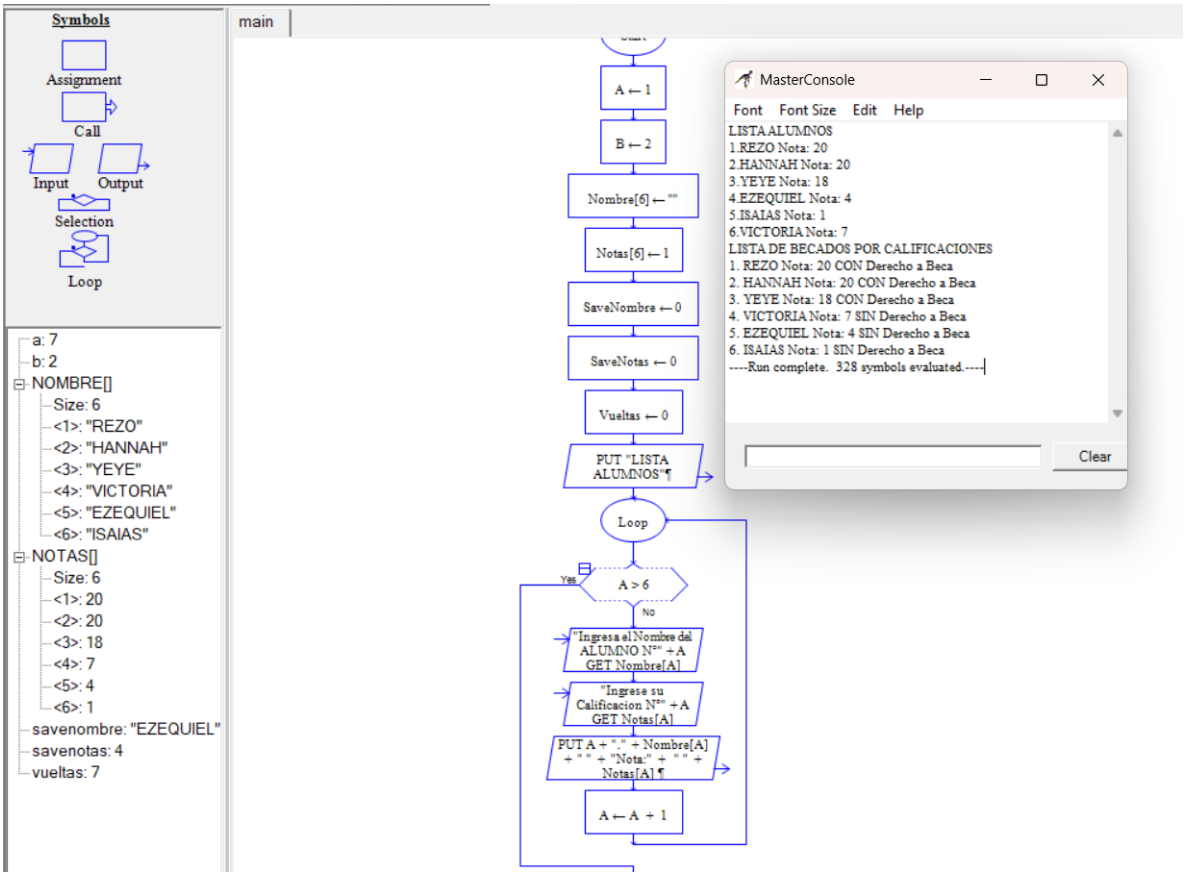
NO CALIFICA

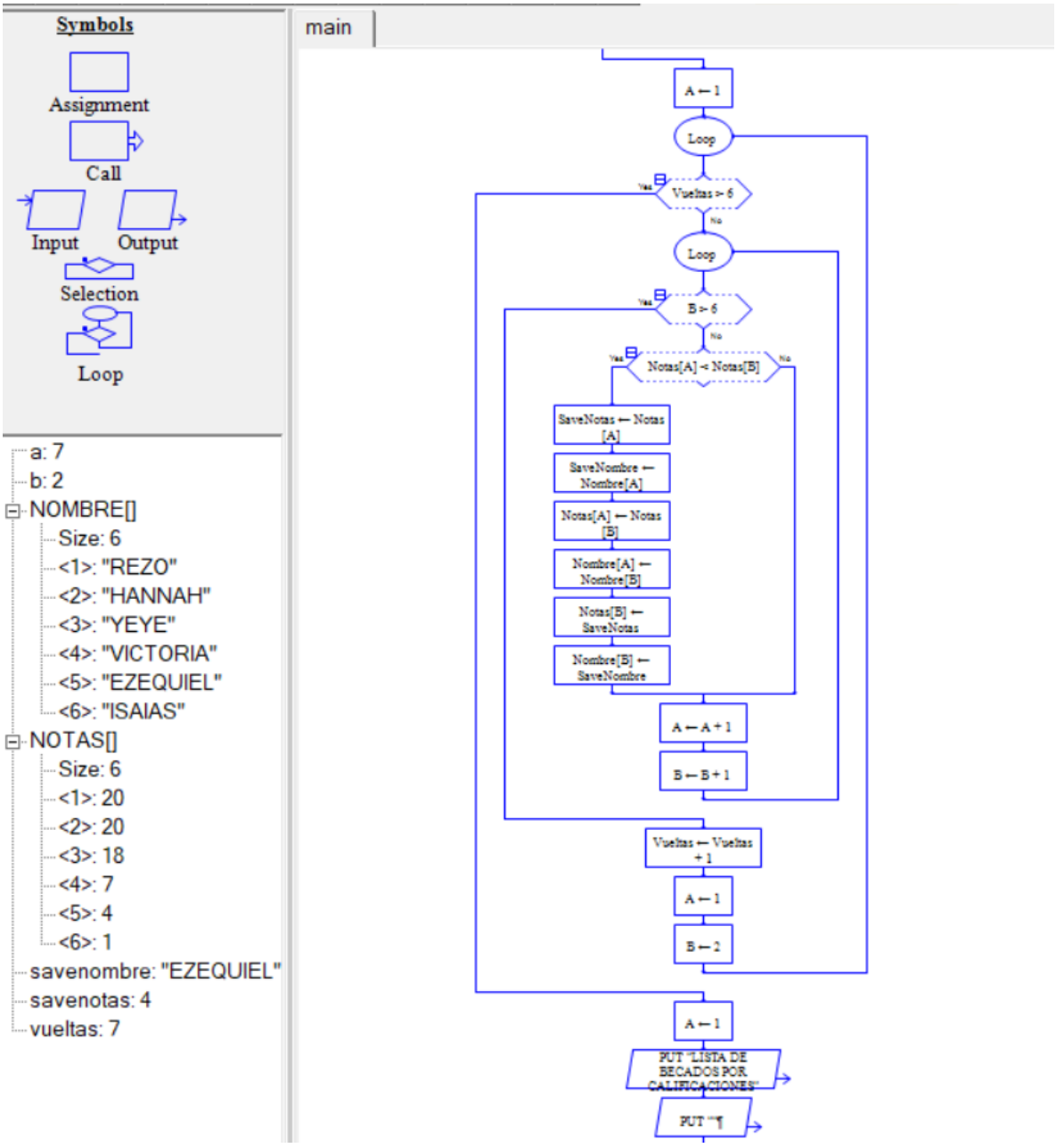
Y MOVEMOS NUESTRA POSICION HASTA RECORRER TODAS LAS **POSICIONES A>6**

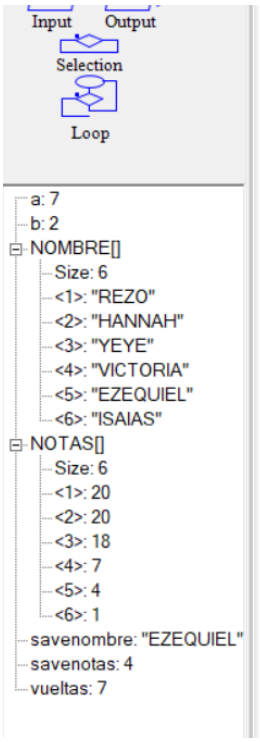
A= A+1

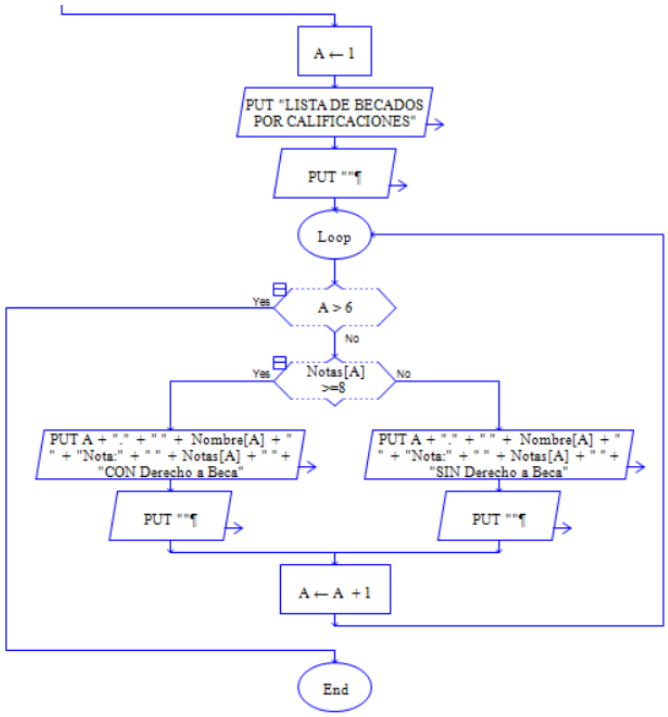
UNA VEZ RECORRIDO LA LISTA IMPRIMIMOS Y LISTO

EXAMPLE:




Symbols
Assignment
Call
Input
Output
Selection
Loop
main
a: 7
b: 2
NOMBRE[]
Size: 6
<1>: "REZO"
<2>: "HANNAH"
<3>: "YEYE"
<4>: "VICTORIA"
<5>: "EZEQUIEL"
<6>: "ISAIAS"
NOTAS[]
Size: 6
<1>: 20
<2>: 20
<3>: 18
<4>: 7
<5>: 4
<6>: 1
savenombre: "EZEQUIEL"
savenotas: 4
vueltas: 7
A ← 1
Loop
Vueltas > 6
Yes
No
Loop
B > 6
Yes
No
Notas[A] < Notas[B]
Yes
No
SaveNotas ← Notas [A]
SaveNombre ← Nombre[A]
Notas[A] ← Notas [B]
Nombre[A] ← Nombre[B]
Notas[B] ← SaveNotas
Nombre[B] ← SaveNombre
A ← A + 1
B ← B + 1
Vueltas ← Vueltas + 1
A ← 1
B ← 2
A ← 1
PUT "LISTA DE BECADOS POR CALIFICACIONES"
PUT ""¶





14. casting

En programación, un casting (o conversión de tipos) es el proceso de transformar una variable de un tipo de dato a otro tipo de dato. Esto se hace para permitir que diferentes tipos de datos interactúen o para realizar ciertas operaciones que requieren un tipo específico.

15. Algoritmos de búsqueda

Un algoritmo de búsqueda es un conjunto de instrucciones diseñadas para encontrar un elemento específico dentro de un conjunto de datos.

REGLAS:

1. El arreglo debe de tener Valores Únicos, es decir, no se deben repetir
2. El arreglo debe estar ordenado de manera ascendente, de menor a mayor

Nota: Para poder Programar el algoritmo de búsqueda binaria es necesario tener 3 apuntadores y saber utilizar un CASTING en programación.

ANALISIS PARA IMPLEMENTARLO

una vez tenido el arreglo definimos lo siguiente:

1. para poder ordenar los elementos utilizamos el método de la burbuja
2. necesitamos 3 posicionadores o apuntadores:

CONDICIONES DE LOS APUNTADORES AL INICIO:

SIEMPRE EL PRIMER APUNTADOR APUNTA A LA POSCION 1 DEL ARRAY

SIEMPRE EL SEGUNDO APUNTADOR APUNTA A LA ULTIMA POSCION DE NUESTRO ARRAY

EL TERCER PUNTADOR VA SER LA SUMA DE LOS DOS APUNTADORES ANTERIORES ENTRE DOS

ES DECIR: (A+B)/2 Y EL RESULTADO DE ESA OPERACION SE LE APLICA UN CATING

A=1 // EL PRIMER APUNTADOR APUNTA A LA POSCION 1 DEL ARRAY

B = //**ULTIMA POSICION ARRAY**

C = CATING(A+B)/2 // ESTE SERA NUESTRO APUNTADOR PUNTO MEDIO

1. necesitamos el valor de búsqueda **busquedavalor** (input)

Una vez definido entramos en las siguientes condiciones anidadas para aplicar la búsqueda:

busquedavalor == C

valor buscado == apuntador punto medio, si es si, entonces termina el programa

sí es no

aplica las siguientes condiciones:

busquedavalor > C

valor buscado > apuntador punto medio, si es si, entonces

A = C+1

primer apuntador = posicion apuntador punto medio + 1

sí es no

busquedavalor < C

valor buscado < apuntador punto medio, si es si, entonces

B = C -1

segundo apuntador = apuntador punto medio -1

y contamos una vuelta

vuelta = vuelta + 1 y comenzamos de nuevo el bucle hasta que si la primera condición se cumple o si vuelta hasta que vueltas > al número de posiciones del array.

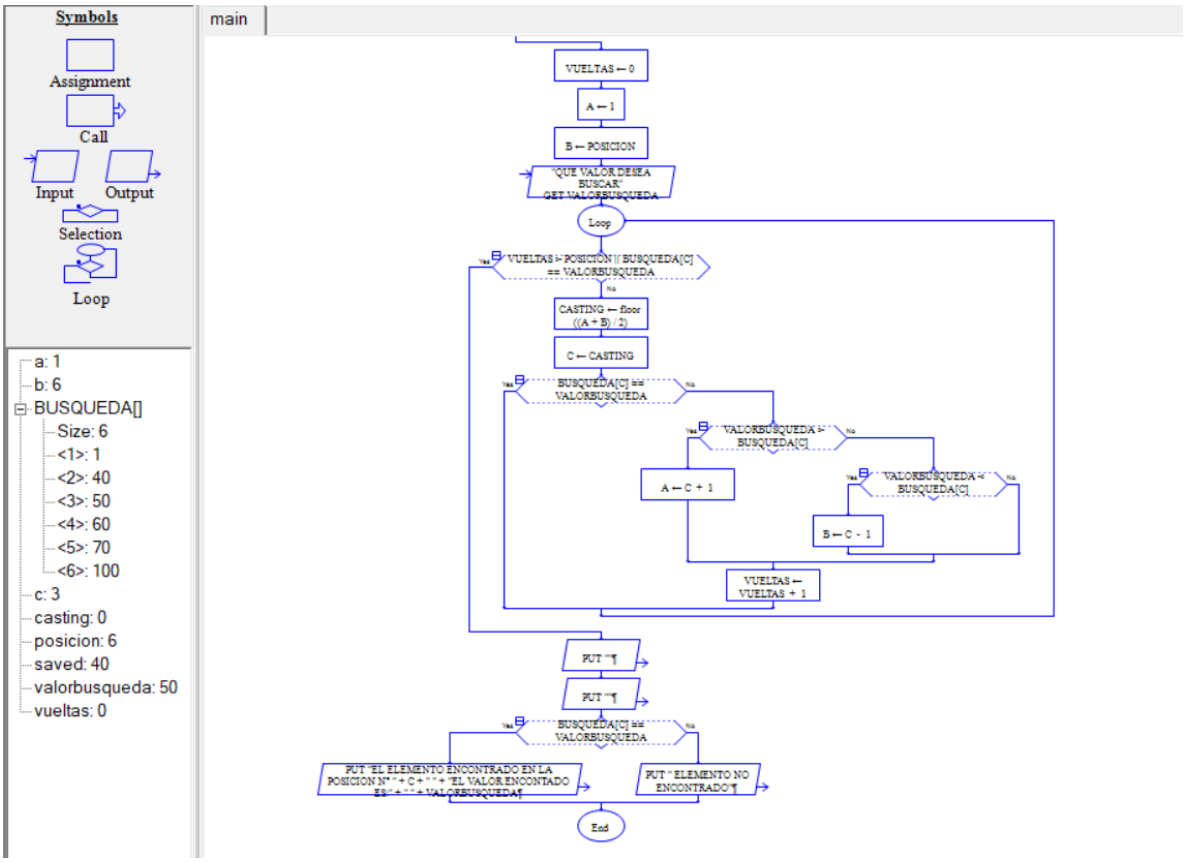
si se cumple la primera condición imprimimos " valor encontrado en la posición N°" + valor punto medio

si no

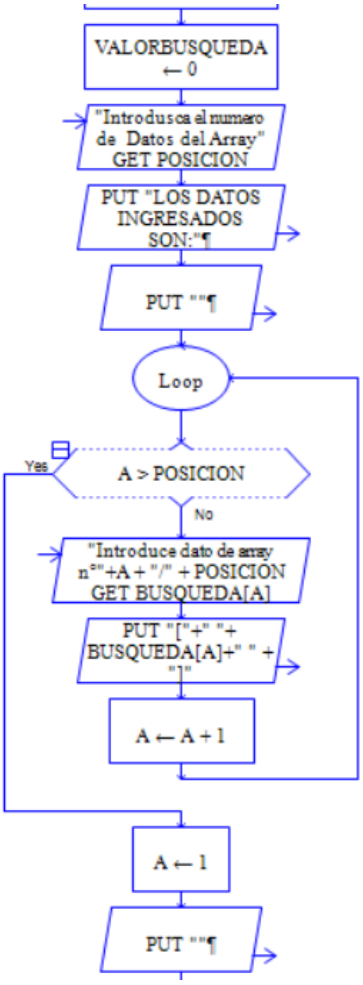
imprimir "valor no encontrado"

Fin

Example:



El siguiente diagrama representa una lista desordenada con n cantidad de números



El siguiente representa una lista ordenada con método burbuja