Problema 1: análisis y relación

1. Explica cómo funciona el programa.
2. Explica en que parte del MAPA DE MEMORIA del proceso se almacena CADA variable usada.
3. Construye un programa similar a este usando Java o C#.
4. Explica en qué parte de la memoria se almacena cada variable de tu programa.
5. Compara ambos programas e indica qué conceptos del programa propuesto se ven reflejados en tu programa.

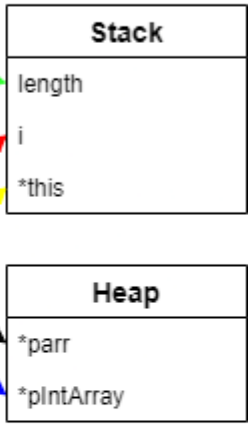
Solución:

1.

- Inicialmente se importan las librerías stdio.h, stdlib.h y string.h.
- Se define la estructura de datos IntArray, dicha estructura tiene dos variables, una es un puntero tipo entero a la dirección del arreglo, y la otra es la variable longitud tipo entero para el arreglo.
- Se declaran todos los métodos/funciones que son necesarias en el programa como lo es crear un arreglo de enteros, iniciar el arreglo, controlar la memoria por medio de liberación, y finalmente imprimir.
- En la función main se hace un llamado a las demás funciones.
- La función createIntArray (int size) reserva en memoria dinamina espacio tanto para la estructura IntArray como para el puntero de esta, también se asigna el tamaño del arreglo y por último la función retorna el puntero al espacio de memoria.
- La función intIntArray (IntArray *this) recibe como parámetro la dirección de memoria de la estructura, luego la recorre y asigna valores del 0 al 99.
- La función printArray (IntArray *this) recorre la estructura ubicada en la posición que recibe por parámetro e imprime cada posición.
- La función destroyIntArray (IntArray *this) libera de la memoria dinámica la estructura de datos y el puntero por medio de free ().

2.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct IntArray{
    int *parr;
    int length;
}IntArray;


IntArray *createIntArray(int size);
void initIntArray(IntArray *);
void destroyIntArray(IntArray *);
void printArray(IntArray *);

int main(){
    IntArray * pIntArray = createIntArray(100);
    initIntArray(pIntArray);
    printArray(pIntArray);
    destroyIntArray(pIntArray);
}

IntArray *createIntArray(int size){
        IntArray *pIntArray = malloc(sizeof(IntArray));
        pIntArray->parr =   malloc( sizeof(int) * size);
        pIntArray->length = size;
        return pIntArray;
}

void initIntArray(IntArray *this){
    for(int i = 0; i < 100; i++){
        this->parr[i] = i;
    }
}

void printArray(IntArray *this){
    for(int i = 0; i < this->length; i++){
        printf("parr[%d]: %d\n", i, this->parr[i]);
    }
}

void destroyIntArray(IntArray *this){
    free(this->parr);
    free(this);
}
```



3. En el repositorio

4.

- parr: heap
- v1: heap
- i: stack
- arr: heap
- length: heap

5. Se puede observar el concepto de punteros, debido que para llenar las posiciones del arreglo se hace por medio de un objeto de una clase que está almacenado en la memoria dinámica, lo mismo pasa con el proceso de impresión. En ambos programas hacemos uso del STACK y del HEAP para las variables e instancias de una clase, solo que en C se hace de forma manual y en C# se hace de forma automática, lo mismo pasa con el proceso de liberar de memoria en C# se administra de forma automática con el GC (Garbage Collector), mientras que en C se administra de forma manual con free () y fflush (). También se puede observar el concepto de malloc () en el momento de crear un objeto con la palabra reservada new ya que se reserva la cantidad de memoria especificada por el parámetro. También podemos observar el uso de estructuras de datos por medio de los objetos en C#.