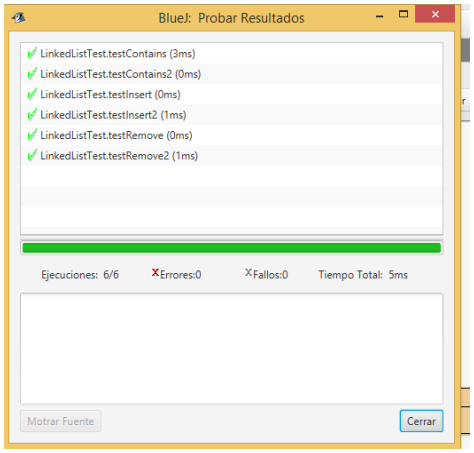
Laboratorio Nro. 4: Implementación de listas enlazadas.

Manuela Valencia Toro
Universidad Eafit
Medellín, Colombia
mvalenciat@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. 

2. **En este ejercicio primero se recibe un entero que indicara el número total de bloques que existirán por así decirlo, con este dato se crea una pila y un arreglo, en cada posición de la pila se almacena el número correspondiente a la posición, al igual que en el arreglo. Seguido de esto, el algoritmo pedirá las instrucciones que se necesiten llevar a cabo, y las desarrollará tal y como fueron descritas en el enunciado, hasta que sea ingresada la palabra quit, la cual hará que la ejecución termine.**

3.
```
public class Main {
    public static int NumberBlocks;
    public static Stack<Integer> Blocks[];
    public static int Position[];
    public static String Line;
    public static int A, B;

    public static void main(String[] args) throws IOException{
        BufferedReader Input = new BufferedReader(new InputStreamReader(System.in));
        NumberBlocks = Integer.parseInt(Input.readLine());
        Blocks = new Stack[NumberBlocks];
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

```
        Position = new int[NumberBlocks];
        for(int i = 0; i < NumberBlocks; i++) {
           Blocks[i] = new Stack<Integer>(); // C+ n
           Blocks[i].push(i); //c + n
           Position[i] = i;  //c + n
        }
         Line = "";
        while(!(Line = Input.readLine()).equals("quit")) {
           StringTokenizer token = new StringTokenizer(Line); // c + n
           String First = token.nextToken();   //c + n
           A = Integer.parseInt(token.nextToken());  //c + n
           String Second = token.nextToken();  //c + n
           B = Integer.parseInt(token.nextToken());  //c + n

           if(A == B || Position[A] == Position[B]) continue; //c + n
           if(First.equals("move")) {  //c + n
              if(Second.equals("onto")) {  //c + n
                 MoveOnto(A, B);  //c + n*m
              } else if(Second.equals("over")) {
                 MoveOver(A, B); //c + n *m
              }
           } else if(First.equals("pile")) {
              if(Second.equals("onto")) {
                 PileOnto(A, B);  //c + n * m
              } else if(Second.equals("over")) {
                 PileOver(A, B); //c + n*m
              }
           }
        }
        for(int i = 0; i < Blocks.length; i++)
System.out.println(Solve(i)); //c + n*m
     }
     public static void MoveOnto(int First, int Second) {
       ClearAbove(Second);
       MoveOver(First, Second);
     }//O(n)
     public static void MoveOver(int First, int Second) {
       ClearAbove(First);
       Blocks[Position[Second]].push(Blocks[Position[First]].pop());
       Position[First] = Position[Second];
     }// O(n)
     public static void PileOnto(int First, int Second) {
       ClearAbove(Second);
       PileOver(First, Second);
     } //O(n)
     public static void PileOver(int First, int Second) {
       Stack<Integer> Pile = new Stack<Integer>();
       while(Blocks[Position[First]].peek() != First) {
          Pile.push(Blocks[Position[First]].pop()); // c + n
       }
       Pile.push(Blocks[Position[First]].pop()); // c
       while(!Pile.isEmpty()) {
         int Tmp = Pile.pop();  //c + n
         Blocks[Position[Second]].push(Tmp); //c + n
         Position[Tmp] = Position[Second]; //c + n
       }
     } //O(n)
     public static void ClearAbove(int Block) {
        while(Blocks[Position[Block]].peek() != Block) {
          Intial(Blocks[Position[Block]].pop());
        } O(n)
     }
```

```
        public static void Intial(int Block) {
           while(!Blocks[Block].isEmpty()) {
             Intial(Blocks[Block].pop());
           }
           Blocks[Block].push(Block);
           Position[Block] = Block;
        }//O(n)
        public static String Solve(int Index) {
           String Result = "";
           while(!Blocks[Index].isEmpty()) Result = " " + Blocks[Index].pop() + Result;
           Result = Index + ":" + Result;
           return Result; //O(n)
}
```

Complejidad= O(m*n)

4. ***en esta complejidad la m representa la complejidad de cada uno de los métodos que se utilizan para que el algoritmo pueda funcionar, ya que cada uno de estos métodos tiene una complejidad O(n), cuando fueron llamados desde un ciclo se puso la m en representación a ese otro ciclo que debe recorrerse cuando hace dicho llamado. Y la n representa el número de instrucciones que sean ingresadas antes de escribir el quit.***

4) Simulacro de Parcial

1. *A) lista.size()>0*
   B. lista.add(auxiliar.pop());
2. *A) auxiliar1.size()>0*
   auxiliar2.size()>0
   B)personas.offer(edad);
3. *c*

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co