**Ejercicios BG**

10oct18 > **14oct18**, empezando en clase

Estos cuatro (4) ejercicios se realizaran en equipo de trabajo, empezando durante la clase, donde el profesor los calificara parcialmente.

Deben usar programación estructurada en la solución de estas preguntas problemas.

El entregable, en archivo “.zip”, deberán subirlo a Comunidad a más tardar el lunes 14 de octubre, incluyendo:

* el archivo reporte (.docx o .pdf), donde en cada pregunta-problema deberán contestar lo que se pregunta, además de que para las tres primeras preguntas problemas deben incluir:
  + un “pantallazo” de la consola de ejecución, y
  + la imagen del diagrama de flujo respectivo,
* los tres (3) archivos de los programas “.asm”, con los nombres de archivo declarados

**PREGUNTAS PROBLEMAS.**

1. Elabore el programa ensamblador “ej1.asm”, que dado un valor n entero, calcule e imprima la siguiente serie sumatoria de n términos:

-1 + 3 - 6 + 10 - 15 + 21 – 28 + *continúa*

donde n es un dato entero >= 1.

Deberán revisar que la N sea correcta, en caso contrario deberá volver a pedir el dato, imprimiendo antes un texto de “ERROR“, y el valor entero tecleado.

Descubra el paso correspondiente entre términos, explíquelo:

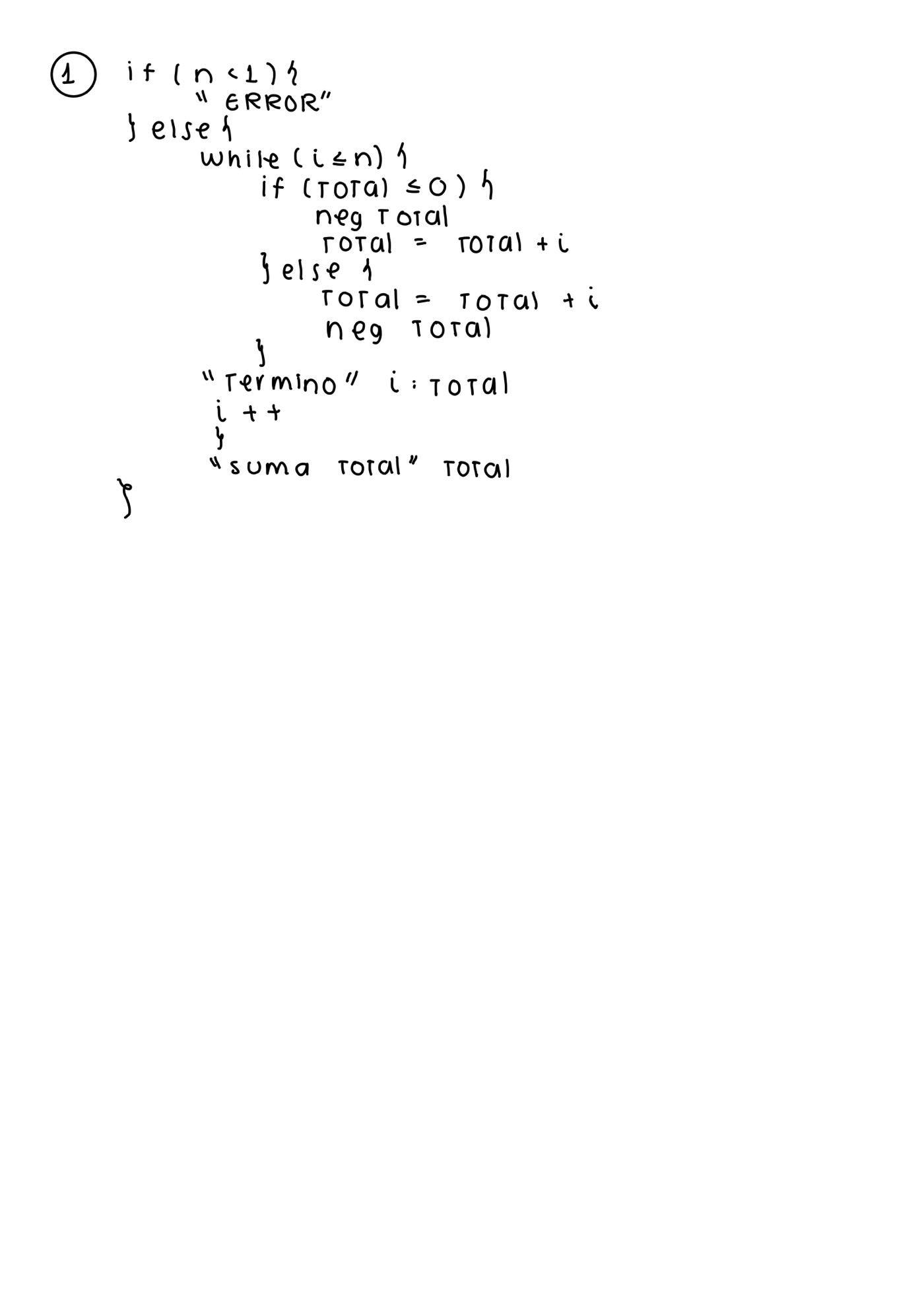
El n-ésimo término de la serie es sumarle *n* al valor absoluto del término *n-1* y multiplicarlo por (-1)^n. Es decir, el n-ésimo término es la suma de los números de 1 a *n*, y va alternando signos.

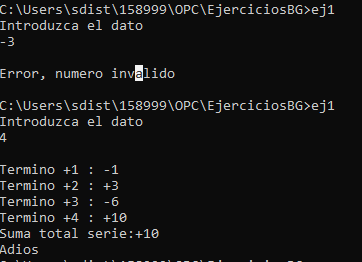
Deberán imprimir los n términos, así como la suma resultante de la serie, con textos adecuados, usando los procedimientos de librería vistos en clase.

El siguiente recuadro despliega el caso de N=4, después de un error.

|  |
| --- |
| Teclee el dato N: -3  ERROR: -3  Teclee el dato N: 4  Termino [+1]: -1  Termino [+2]: +3  Termino [+3]: -6  Termino [+4]: +10  SUMA TOTAL SERIE: +6  ADIOS. |

HINT: Para cambio de signo en los términos, puede usar IMUL, y / o NEG.





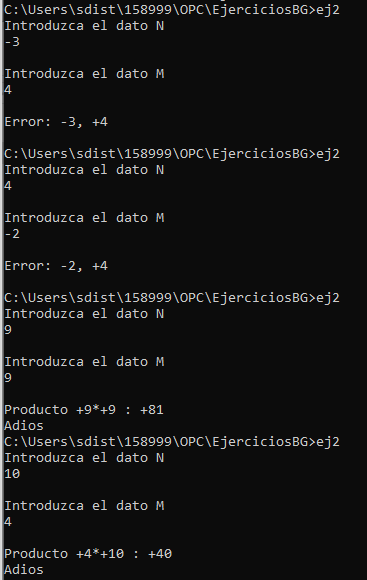
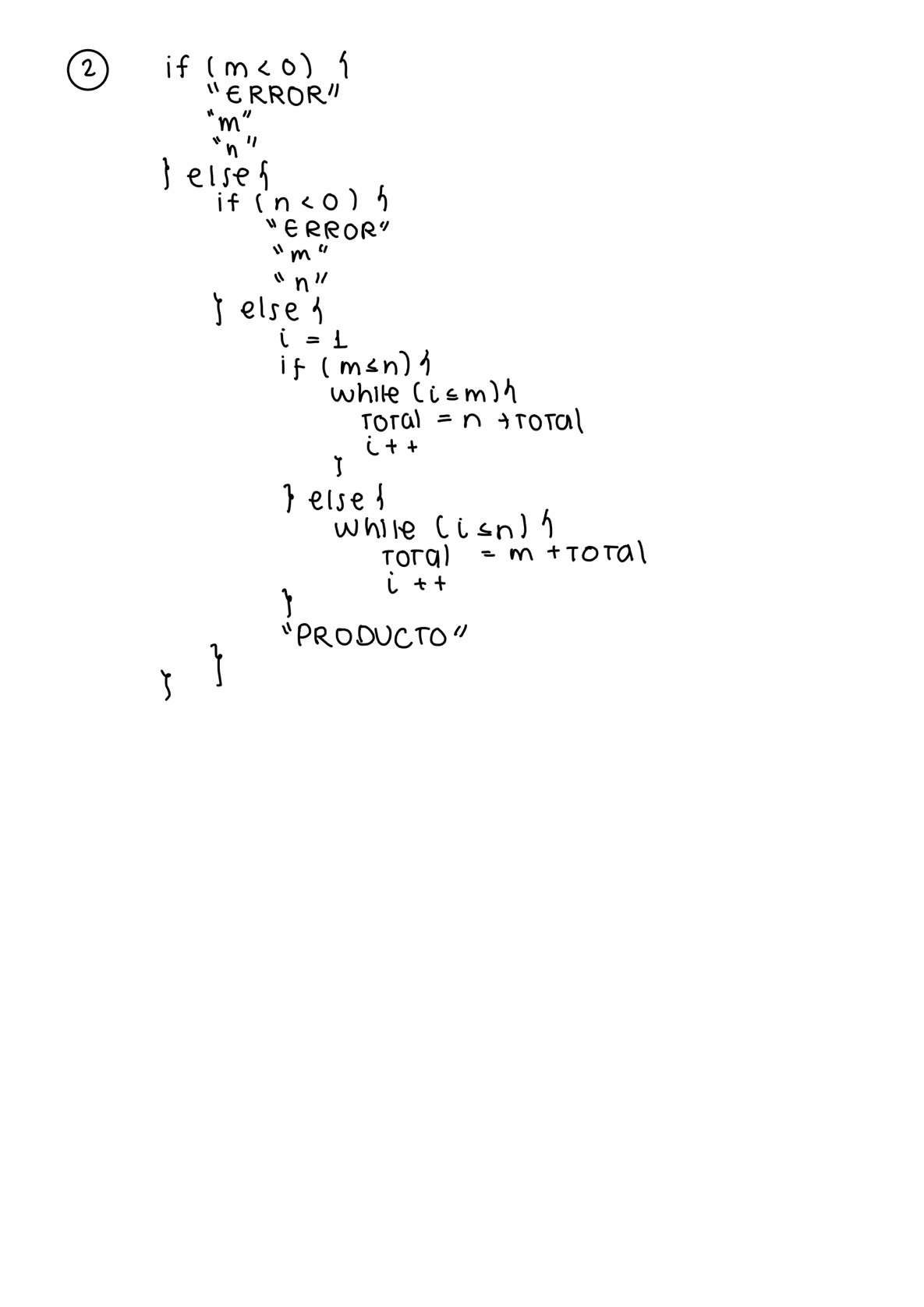
1. Elabore el programa ensamblador “ej2.asm”, y revise que los datos dados a m y n sean m ≥ 0 y n ≥ 0, (ambos datos enteros) para calcular el producto m\*n sin utilizar la operación de multiplicación. En caso de no cumplirse la condición anterior el programa deberá terminar imprimiendo un texto de error e imprimiendo los valores de m y n tecleados inicialmente.

Haga los movimientos necesarios para que independientemente de cómo se hayan teclados los datos, m<=n se cumpla, y calcule el producto haciendo repetir m veces n, la forma más óptima.

Deberá imprimir el producto resultante, con el texto correspondiente, como se muestra en las dos alternativas de despliegue, según sea el caso.

|  |
| --- |
| Teclee el dato M: 4  Teclee el dato N: -3  ERROR: +4, -3  ADIOS. |

|  |
| --- |
| Teclee el dato M: 4  Teclee el dato N: 3  PRODUCTO (+3)\*(+4): +12  ADIOS. |



1. Elabore el programa ensamblador “ej3.asm”, que calcule factorial de n (n!), recordando que para n! con:

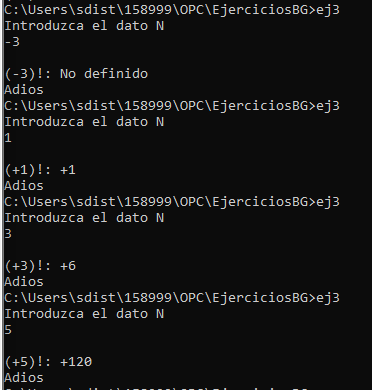
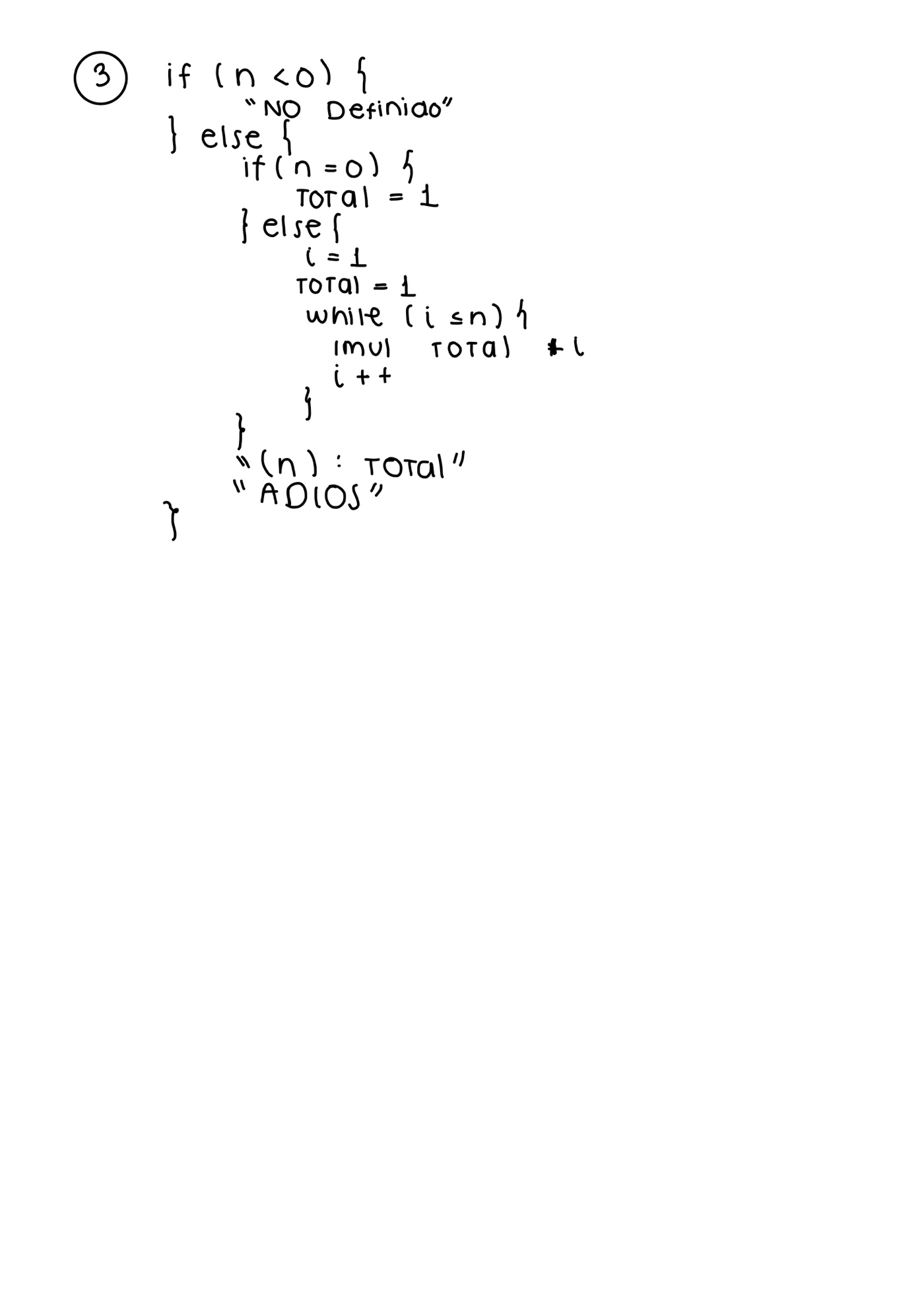
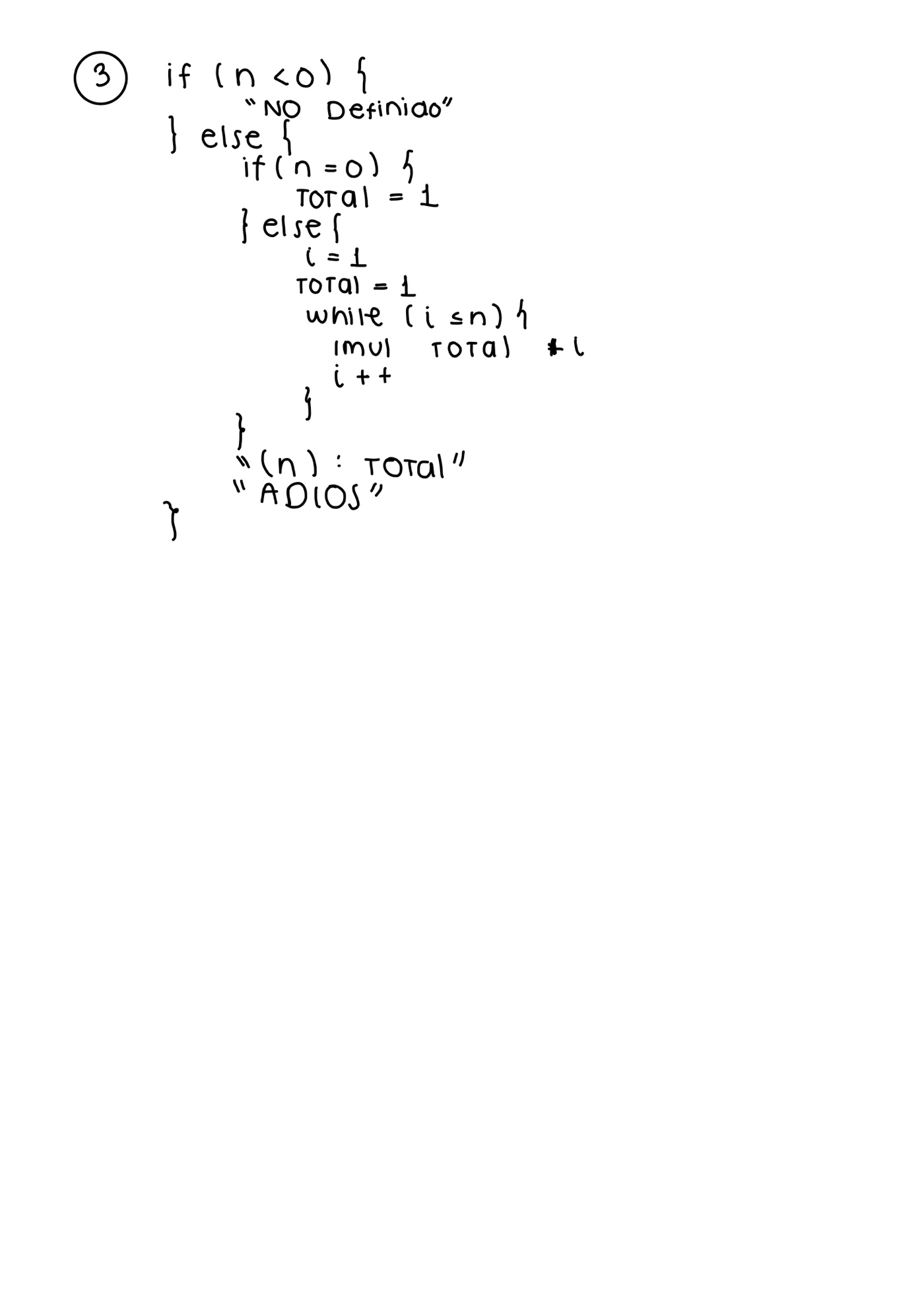
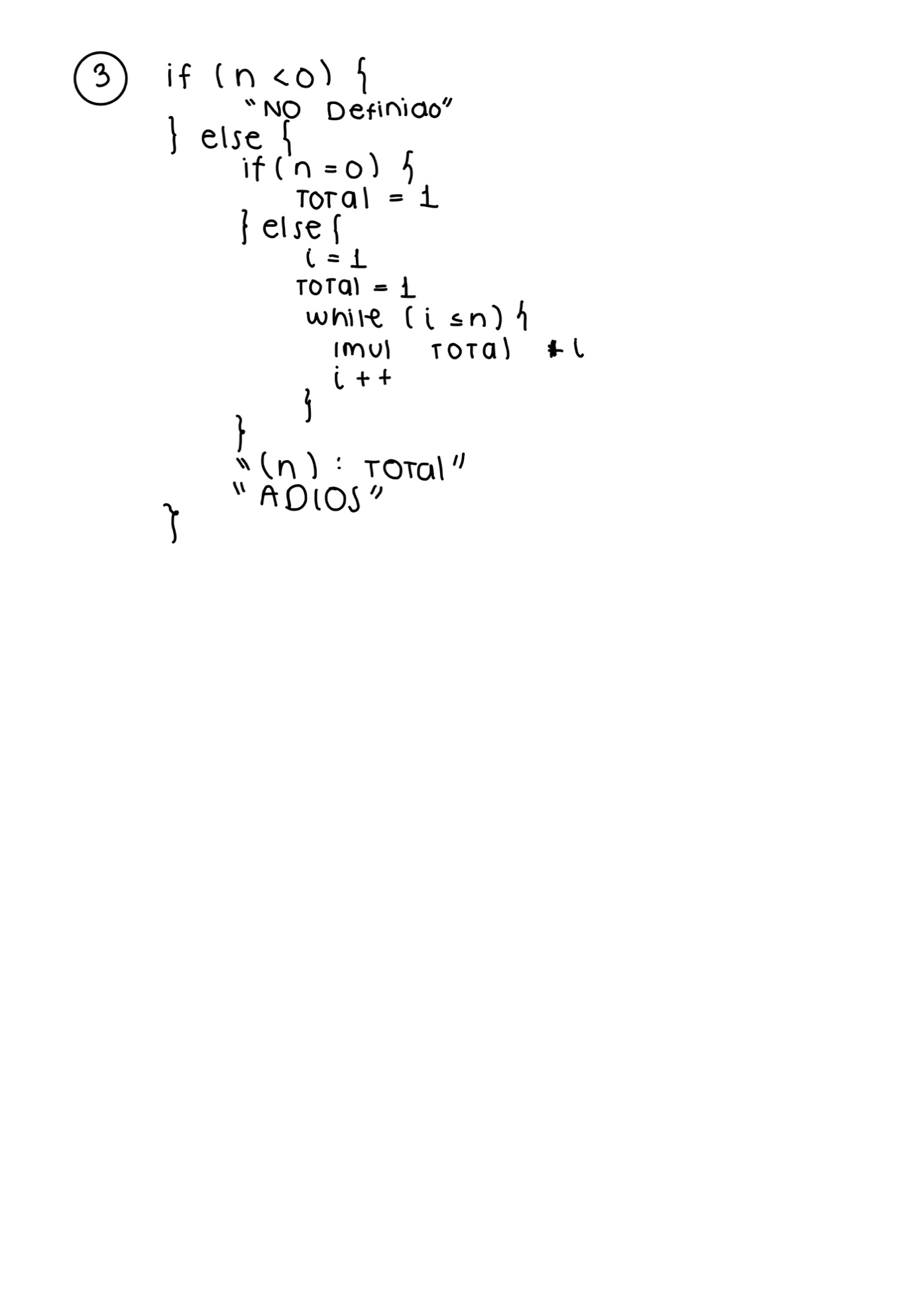
* n < 0, n! no está definido,
* n = 0, n! = 1, y
* n > 0, n! está definido como 1\*2\*3\*4\*…\*(n-2)\*(n-1)\*n

Despliegue según corresponda:

|  |
| --- |
| Teclee el dato N: 4  (+4!): +24  ADIOS. |

|  |
| --- |
| Teclee el dato N: -3  (-3!): No definido  ADIOS. |

|  |
| --- |
| Teclee el dato N: 0  (+0!): +1  ADIOS. |



1. Estudie la presentación “matBG.pptx” sobre las instrucciones bitwise:

* AND,
* OR,
* XOR,
* NOT, y
* TEST.

Y conteste a las siguientes preguntas:

1. ¿Qué significa que en el Lenguaje Ensamblador de x86 no existan instrucciones de lógica Booleana? ¿Cómo son las instrucciones de lógica Booleana?

No hay instrucción que permita evaluar múltiples condiciones al mismo tiempo. El procesador no maneja el concepto de “True” y “False”, lo único que se tiene para evaluar una condición son las banderas de estado, dentro de las cuales no existen las banderas True/False.

1. ¿Qué significa que en el Lenguaje Ensamblador de x86 existan instrucciones Booleanas bitwise?

Las instrucciones bit a bit no requieren un almacenamiento de resultados con banderas. Las operaciones booleanas bit a bit solo requieren de una tabla de verdad que identifica a la operación. Todo se puede implementar a nivel hardware.

1. Traduzca a Lenguage Ensamblador de x86, más abajo, el siguiente trozo de programa de algún Lenguaje de programación de Alto Nivel.

Suponga que las variables A, B y C son enteros a 4 bytes.

|  |
| --- |
| IF( (A > B) && (C <= 5) ) {  C = B – A;  } else {  C = B \* 10;  }  cout << “C: “ << C << “\n” |

A continuación su código traducido a Lenguaje Ensamblador de x86, lo más óptimo posible:

|  |
| --- |
| ; Aquí va la traducción |

1. LECTURA Y ESTUDIO dos capítulos del libro CODE:

* Capítulo 9, “Bit by Bit by Bit”.
* Capítulo 10, “Logic and Switches”