UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Programación II
Práctica Calificada 1
Pregrado
2022-II
Profesor José Fiestas
Lab 1.03

Indicaciones específicas:

- Esta evaluación contiene 8 páginas (incluyendo esta página) con 3 preguntas. El total de puntos son 20.
- El tiempo límite para la evaluación es 100 minutos.
- Cada pregunta deberá ser respondida en un solo archivo con el número de la pregunta.
  - `p1.cpp`
  - `p2.cpp`
  - `p3.cpp`
- Deberás subir estos archivos directamente a www.gradescope.com, uno en cada ejercicio. También puedes crear un .zip
- La evaluación es **individual**. Un nivel importante de **similitud** con otros estudiantes o fuentes externas no será aceptada y se **anulará** el ejercicio. Si se utiliza una fuente externa parcial para alguna función o parte del ejercicio, deberá **hacer referencia a ella en la entrega**, y se considerará justificada, pero solo se calificará lo desarrollado por el alumno.
- Se puede **consultar material de clase** y utilizar funciones o partes de código desarrollados en clase. Esto ultimo no descontará puntos, pero también se debe **hacer referencia a ellos en la entrega**, (de no mencionarlo, no se podrá saber si utilizó una fuente de clase o externa).

Competencias:

- Para los alumnos de la carrera de Ciencia de la Computación

  Aplicar conocimientos de computación y de matemáticas apropiadas para la disciplina. (Evaluar)

  Analizar problemas e identificar y definir los requerimientos computacionales apropiados para su solución.(Usar)

  Utilizar técnicas y herramientas actuales necesarias para la práctica de la computación. (Usar)

- Para los alumnos de las carreras de Ingeniería

  Capacidad de aplicar conocimientos de matemáticas (nivel 3)

  Capacidad de aplicar conocimientos de ingeniería(nivel 2)

  Capacidad para diseñar un sistema, un componente o un proceso para satisfacer las necesidades deseadas dentro de restricciones realistas (nivel 2)

Calificación:

Tabla de puntos (sólo para uso del professor)

Question	Points	Score
1	6	
2	7	
3	7	
Total:	20	

1. (6 points) **¡Adivinen mi número!**

Paolo está con 4 amigos. Él piensa un número entre 1 y 50, y dice a sus amigos: '¡Adivinen mi número!'. Sus amigos deben escribir el número en un papel y mostrarlo simultáneamente. Ninguno de ellos puede ver el número del otro antes de mostrar su papel.

Paolo les da 1000 intentos para hacerlo. Si logran todos adivinar el número en el mismo intento, Paolo los invita a cenar al Restaurante Central (el más caro de Lima). Paolo sabe algo de estadística y se da cuenta que es difícil adivinar bajo esas condiciones. Para programarlo, afortunadamente, no hay que saber de Estadística.

Desarrolle un código que haga este experimento e indique en qué intento, la mayor cantidad de amigos lograron adivinar el mismo número. Quizás no sean los 4, pero con algo de suerte, 2 o 3

Por ejemplo, una salida del código podrían ser:

```
Paolo piensa en el numero 12
Maximo 2 han pensado al mismo tiempo el mismo numero en el
   intento 449
```

Entonces, sus amigos se quejan y piden más intentos. Paolo ve en riesgo su estrategia e incrementa el número de intentos a 100000. Indique qué observa en ese caso ¿Tendrá que invitar Paolo a sus amigos?

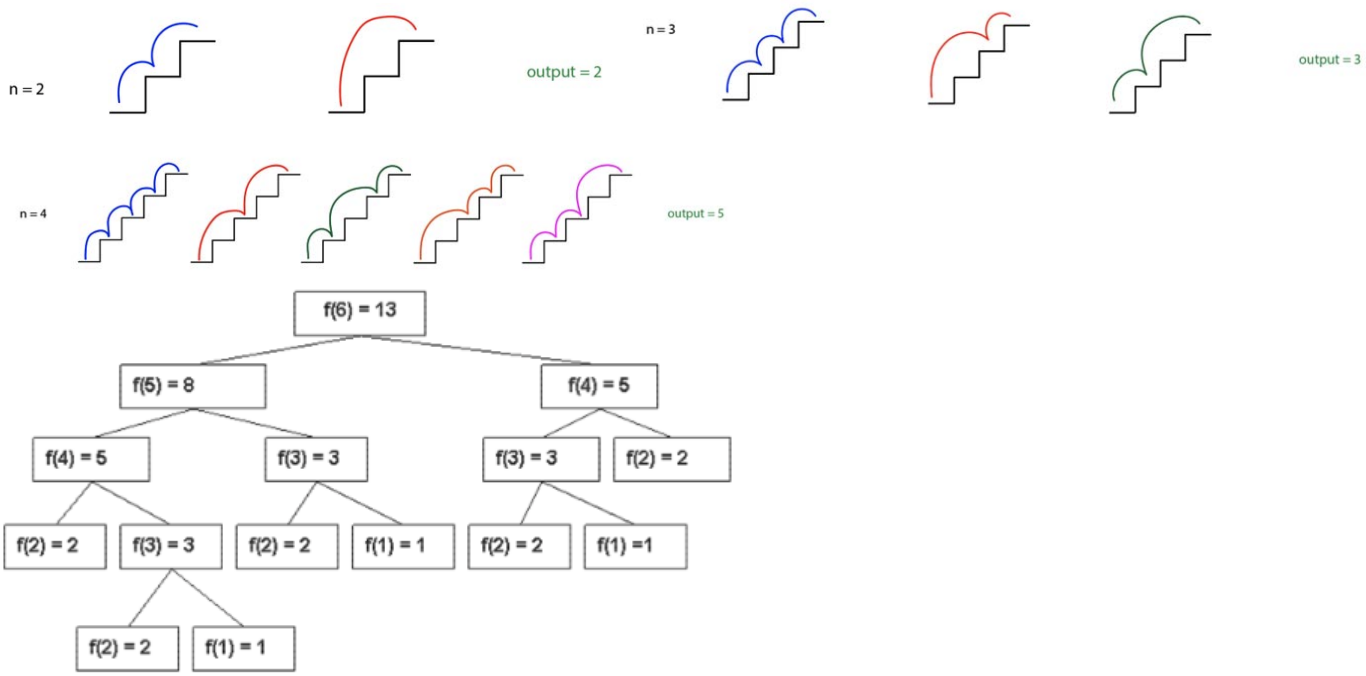
La rúbrica para esta pregunta es:

Criterio	Excelente	Adecuado	Mínimo	Insuficiente
Ejecución: Evalúa el diseño del algoritmo, siguiendo buenas prácticas en programación. Asi como la ejecución del mismo	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (`3pts`)	El diseño del algoritmo es ordenado y claro, pero optimizable. La ejecución es correcta (`2pts`)	El diseño del algoritmo no es ordenado ni claro. La ejecución es correcta (`1pts`).	El diseño del algoritmo no es ordenado ni claro. La ejecución no es correcta (`0.5pts`)
Sintáxis: Evalúa sintaxis en el código y correcta ejecución (semántica)	No existen errores sintácticos o de compilación. La ejecución es correcta (`2pts`)	Existen algunos errores sintácticos, que no afectan directamente el resultado, pero hacen al código optimizable. (`1.5pts`).	Existen errores sintácticos o de ejecución, que afectan parcialmente el resultado (`1pts`).	El código tiene errores de sintáxis y de ejecución que no permiten obtener un resultado correcto (`0.5pts`).
Optimización: evalua uso de buenas practicas en programación en el diseño del algoritmo y el código de programación, para lograr un nivel de eficiencia adecuado	El código es óptimo y eficiente. De buen performance e interacción con el usuario (`1pt`)	El codigo es de buen performance durante la ejecución pero optimizable. Pero no afecta el resultado. (`0.8pts`).	El código no esta optimizado, lo que afecta parcialmente el resultado. (`0.5pts`).	El codigo no esta optimizado y la ejecución es deficiente (incluye la no entrega del ejercicio) (`0pts`).

2. (7 points) **Subir escaleras**
Implemente la solución del problema de encontrar las posibles formas de subir una escalera de n peldaños, si se pueden solo subir 1 o 2 peldaños a la vez.

Aquí la fórmula:

Se puede demostrar que la solución de la cantidad de formas posibles de subir n peldaños es igual a la suma de la cantidad de formas de subir (n-1) peldaños + la cantidad de formas de subir (n-2) peldaños. Las figuras muestran las formas de subir escaleras de n=2,3,4 peldaños, y el diagrama, describe la forma de cálculo para n=6



Programe la solución en forma **recursiva** o **no-recursiva** (a elección del estudiante) para calcular las formas de subir escaleras. Pruebe el algoritmo para n= 10, 25, y 45 peldaños. ¿Encuentra algún problema al calcular un número de peldaños mayor que 50? ¿En ese caso, qué solución propone? (no necesita programarla, solo comentarla)

Ejemplos:

```
cantidad de peldaños: 10
Tiene 89 posibilidades de subir una escalera con 10 peldaños
```

```
cantidad de peldaños: 25
Tiene 121393 posibilidades de subir una escalera con 25 peldaños
```

La rúbrica para esta pregunta es:

Criterio	**Excelente**	**Adecuado**	**Mínimo**	**Insuficiente**
Ejecución: Evalúa el diseño del algoritmo, siguiendo buenas prácticas en programación. Asi como la ejecución del mismo	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (`3pts`)	El diseño del algoritmo es ordenado y claro, pero optimizable. La ejecución es correcta (`2pts`)	El diseño del algoritmo no es ordenado ni claro. La ejecución es correcta (`1pts`).	El diseño del algoritmo no es ordenado ni claro. La ejecución no es correcta (`0.5pts`)
Sintáxis: Evalúa sintaxis en el código y correcta ejecución (semántica)	No existen errores sintácticos o de compilación. La ejecución es correcta (`2pts`)	Existen algunos errores sintácticos, que no afectan directamente el resultado, pero hacen al código optimizable. (`1.5pts`).	Existen errores sintácticos o de ejecución, que afectan parcialmente el resultado (`1pts`).	El código tiene errores de sintáxis y de ejecución que no permiten obtener un resultado correcto (`0.5pts`).
Optimización: evalua uso de buenas practicas en programación en el diseño del algoritmo y el código de programación, para lograr un nivel de eficiencia adecuado	El código es óptimo y eficiente. De buen performance e interacción con el usuario (`2pts`)	El codigo es de buen performance durante la ejecución pero optimizable. Pero no afecta el resultado. (`1.5pts`).	El código no esta optimizado, lo que afecta parcialmente el resultado. (`1pts`).	El codigo no esta optimizado y la ejecución es deficiente (incluye la no entrega del ejercicio) (`0pts`).

3. (7 points) **Intercambio de valores en arrays**
Resuelva el problema de intercambiar los valores de dos arrays de tamaño n, de la siguiente manera

Si **n es par**:

- Se intercambian los valores de los elementos con índice **i par, del primer array** con los siguientes elementos **(i+1) del segundo array**
- Se intercambian los valores de los elementos con índice**i impar, del primer array** con los anteriores elementos **(i-1) del segundo array**

Si **n es impar**:

- Se sigue el mismo procedimiento que en el caso anterior, pero con excepción del ultimo elemento del primer array (ya que no existirá un siguiente elemento en el segundo array)
- En este caso, el ultimo elemento del primer array (i=n-1), se intercambia directamente con el último elemento del segundo array

Importante:

- **Utilice punteros estáticos a arrays**
- **El main() debe contener principalmente llamadas a funciones, con excepción de la declaración de los arrays estáticos**
- **Recuerde que un array de 5 elementos (n=5) tiene índices i=0,1,2,3,4, es decir, en ese caso, los pares serian 0,2,4, y los impares 1,3**

Ejemplos:

```
Ingrese tamaño de los arrays:
5
Primer array: 13 15 19 24 16
Segundo array: 18 10 24 9 15
Primer array modificado: 10 18 9 24 15
Segundo array modificado: 15 13 24 19 16
```

```
Ingrese tamaño de los arrays:
8
Primer array: 13 15 19 24 16 18 10 24
Segundo array: 9 15 6 11 18 8 13 9
Primer array modificado: 15 9 11 6 8 18 9 13
Segundo array modificado: 15 13 24 19 18 16 24 10
```

La rúbrica para esta pregunta es:

Criterio	**Excelente**	**Adecuado**	**Mínimo**	**Insuficiente**
Ejecución: Evalúa el diseño del algoritmo, siguiendo buenas prácticas en programación. Asi como la ejecución del mismo	El diseño del algoritmo es ordenado y claro, siguiendo buenas prácticas en programación. La ejecución es correcta (`3pts`)	El diseño del algoritmo es ordenado y claro, pero optimizable. La ejecución es correcta (`2pts`)	El diseño del algoritmo no es ordenado ni claro. La ejecución es correcta (`1pts`).	El diseño del algoritmo no es ordenado ni claro. La ejecución no es correcta (`0.5pts`)
Sintáxis: Evalúa sintaxis en el código y correcta ejecución (semántica)	No existen errores sintácticos o de compilación. La ejecución es correcta (`2pts`)	Existen algunos errores sintácticos, que no afectan directamente el resultado, pero hacen al código optimizable. (`1.5pts`).	Existen errores sintácticos o de ejecución, que afectan parcialmente el resultado (`1pts`).	El código tiene errores de sintáxis y de ejecución que no permiten obtener un resultado correcto (`0.5pts`).
Optimización: evalua uso de buenas practicas en programación en el diseño del algoritmo y el código de programación, para lograr un nivel de eficiencia adecuado	El código es óptimo y eficiente. De buen performance e interacción con el usuario (`2pts`)	El codigo es de buen performance durante la ejecución pero optimizable. Pero no afecta el resultado. (`1.5pts`).	El código no esta optimizado, lo que afecta parcialmente el resultado. (`1pts`).	El codigo no esta optimizado y la ejecución es deficiente (incluye la no entrega del ejercicio) (`0pts`).