UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

# Programación II
## Práctica Calificada 1
## Pregrado
## 2023-II
## Profesor José Fiestas
## Lab 2.01

## 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**. **Similitud** con otras entregas 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 con un comentario en el código**, 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

  Analizar un problema computacional complejo y aplicar principios de computación y otras disciplinas relevantes para identificar soluciones.

  Diseñar, implementar y evaluar una solución computacional para satisfacer un conjunto determinado de requerimientos computacionales en el con- texto de la disciplina del programa.

- Para los alumnos de las carreras de Ingeniería

  La capacidad de identificar, formular y resolver problemas complejos de ingeniería mediante la aplicación de principios de ingeniería, ciencias y matemáticas.

- Para Administración y Negocios Digitales

  Pensamiento analítico y crítico: Organizar, analizar e interpretar información para tomar decisiones y orientar su propia conducta.

  Fundamentos de tecnología: Comprender las principales tendencias en tecnología y su potencial, así como de adoptar y utilizar tecnologías para la solución de problemas.

---

## Calificación:

Tabla de puntos (sólo para uso del professor)

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

1. (6 puntos) **Suma de cubos**

   Desarrolle un código en C++ que demuestre la siguiente sumatoria

$$\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2$$

   Es decir, $1^3 + 2^3 + 3^3 + ... + (n-1)^3 + n^3 = (\frac{n(n+1)}{2})^2$

   El usuario debe ingresar el valor de n (cantidad total de iteraciones) y calcular la suma. Además,

   - debe calcular el error de aproximación comparando el valor obtenido de la suma con el valor de la fórmula, e imprimirlo.
   - haga pruebas para n=10, 100, 200, 300 e incremente sucesivamente el valor en 100 hasta que el código obtenga un resultado incorrecto.
   - determine para que valor de n obtiene errores y optimice el código, si lo considera posible, para solucionar el error.
   - comente, según lo visto en clase, a que se debe el error, o que podría originar un error al calcular una suma con muchos dígitos.
   - El código debe descartar valores menores o iguales a cero para n.

   Una prueba sería:
   *si n= 51, se obtiene:*
   *suma: 1758276, según la fórmula: 1758276, error: 0*

   **Nota:** Utilice las herramientas de programación vistas en clase. Las instrucciones para el ingreso de datos deben estar claras en el código, así como la descripción de las salidas.

La rúbrica para esta pregunta es:

| **Criterio** | **Excelente** | **Adecuado** | **Mínimo** | **Insuficiente** |
|---|---|---|---|---|
| **Algoritmo**: Evalúa el diseño del algoritmo, asi como la ejecución del mismo | El diseño del algoritmo es ordenado y claro. 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, para lograr un nivel de eficiencia adecuado | El código es óptimo y eficiente. De buen performance e interacción con el usuario (`1pt`) | El código 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 código no esta optimizado y la ejecución es deficiente (incluye la no entrega del ejercicio) (`0pts`). |

2. (7 puntos) **Costo de vuelo**

Determine costos de boletos de vuelo con las siguientes caracteristicas:

- **destino:** Berlin, Madrid, Sidney o Seul
- **tiempo de viaje:** 15, 12, 24, 27 horas respectivamente
- **temporada:** alta o baja
- **clase:** business o economy

Utilice **funciones** para calcular el costo del viaje:

- multiplicando el tiempo de viaje por 50
- multiplicando este resultado por 2 si es temporada alta
- multiplicando este resultado por 4 si es clase business
- si el destino es Madrid, se agregan 500; si es Berlin, 1000; si es Sidney, 1500; y si es Seul, 2000

El código debe realizar lo siguiente:

- pedir los datos de entrada (input)
- calcular costo de 5 boletos de viaje
- calcular los precios e imprimirlos
- el *main()* debe ser claro y breve. Debe consistir solo de la declaración de variables y llamada a funciones.

Una salida del código para un destino podría ser:

```
Ingrese destino: Berlin
Ingrese temporada: alta
Ingrese clase: business
El costo del boleto es: 6500
```

La rúbrica para esta pregunta es:

| **Criterio** | **Excelente** | **Adecuado** | **Mínimo** | **Insuficiente** |
|---|---|---|---|---|
| **Algoritmo**: Evalúa el diseño del algoritmo, asi como la ejecución del mismo | El diseño del algoritmo es ordenado y claro. 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, para lograr un nivel de eficiencia adecuado | El código es óptimo y eficiente. De buen performance e interacción con el usuario (`2pt`) | El código 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. (`1pt`). | El código no esta optimizado y la ejecución es deficiente (incluye la no entrega del ejercicio) (`0pts`). |

3. (7 puntos) **Vía de un carril**

   En una vía de un solo carril, los autos deben esperar a los que están delante de ellos para avanzar. Escriba un código en C++ para simular esta situación, en un carril donde se pueden alinear un máximo de 20 autos. Llene el carril con 7 autos en posiciones aleatorias y permita que los autos avancen, en dos situaciones:

   - si no hay un auto delante de él, avanza
   - si hay un auto delante de él, no avanza

   Desarrolle los siguientes puntos:

   - Aplique este método a cada auto de la vía (avance) y repita el procedimiento hasta que todos los autos lleguen al final del carril. Note que los autos que lleguen al final deben salir de ella.
   - Imprima la ubicación de los autos en cada avance.
   - Determine después de cuántos avances todos los autos recorrieron la vía
   - Utilice funciones y punteros. El main debe ser claro y breve.

   Una salida del código para 5 avances (la primera linea muestra las posiciones iniciales de los autos) sería:

```
1 1 0 0 0 0 1 0 0 1
1 0 1 0 0 0 0 1 0 0
0 1 0 1 0 0 0 0 1 0
0 0 1 0 1 0 0 0 0 1
0 0 0 1 0 1 0 0 0 0
0 0 0 0 1 0 1 0 0 0
```

La rúbrica para esta pregunta es:

| **Criterio** | **Excelente** | **Adecuado** | **Mínimo** | **Insuficiente** |
|---|---|---|---|---|
| **Algoritmo**: Evalúa el diseño del algoritmo, asi como la ejecución del mismo | El diseño del algoritmo es ordenado y claro. 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, para lograr un nivel de eficiencia adecuado | El código es óptimo y eficiente. De buen performance e interacción con el usuario (`2pt`) | El código 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. (`1pt`). | El código no esta optimizado y la ejecución es deficiente (incluye la no entrega del ejercicio) (`0pts`). |