Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 – Estructuras de Datos y Algoritmos

2025 - 1

Tarea 0

Fecha de entrega código: 28 de Marzo, 23:59 Chile continental

Link a generador de repos: CLICK AQUÍ

Objetivos

- Diseñar algoritmos simples en C.
- Comprender las diferencias entre Arrays y Listas ligadas.
- Familiarizarse con el uso de punteros y manejo de memoria.

Introducción

¡Hey Ferb, ya sé qué vamos a hacer hoy!

Debido a tus excepcionales habilidades como programador, Phineas y Ferb han decidido contactarte para una tarea muy especial. Han creado un parque de diversiones con grandes atracciones y montañas rusas, pero necesitan de alguien que lo administre correctamente, para evitar que su proyecto sea un desastre. Phineas y Ferb confían en que tus habilidades pueden llevar su parque al siguiente nivel, ¡y todo depende de ti para que la diversión no se detenga!

Problema

El parque ha abierto sus puertas y el caos ha comenzado. Las montañas rusas no están organizadas correctamente, las filas son interminables y los visitantes comienzan a perder la paciencia. Para empeorar las cosas, los carros tienen una mala distribución de peso, poniendo en riesgo la seguridad de los pasajeros. Si no se soluciona pronto, las vacaciones de todos estarán arruinadas. Tu misión es utilizar estructuras y funciones para optimizar el parque, hacer que las atracciones sean más eficientes y asegurarte de que cada visitante tenga una experiencia inolvidable. ¡Es hora de programar y salvar las vacaciones!

Entidades[1]

- **Montaña Rusa**: Cuenta con una ID[2].
- **Carro**: Cuenta con ID y con una capacidad de P personas.
- **Persona**: Cuenta con ID, estatura y peso.

Primero, recibirás una línea con un entero N, indicando la cantidad máxima de montañas rusas que puede tener el parque de atracciones. Luego, recibirás una línea con un entero E que indica la cantidad de eventos a recibir. A continuación, se te entregarán los E eventos, donde cada uno puede contener más de una línea de input que deberás procesar. Puedes ver ejemplos de eventos en la sección de input.

Parte 1: Preparando el Parque de Diversiones

1. `BUILD {K} {P} {estatura_minima} {peso_maximo}`

Este evento construye una montaña rusa, registrando sus {K} carros, cada uno con capacidad para {P} personas y con un mínimo de {estatura_minima} para subir. También, se recibe un peso máximo, que indica el máximo peso que pueden sumar entre todos los pasajeros de la montaña rusa. Luego, recibe K líneas, donde se entrega el ID de cada carro.

input

```
BUILD {K} {P} {estatura_minima} {peso_maximo}
{id_1}
{id_2}
...
...
{id_K}
```

Imprime la montaña rusa con su ID y su capacidad total[3]. Luego se imprime cada carro con su ID correspondiente:

output

```
Montana rusa creada, de id {ID} y capacidad de {cantidad} personas:
    Carro {id_1}
    Carro {id_2}
    ...
    ...
    Carro {id_K}
```

[1] Los atributos entregados son los mínimos. Puedes agregar todos los que consideres necesarios para resolver la tarea.
[2] La ID de las montañas rusas serán correlativas, enumeradas de 0 en adelante según el orden en que sean construidas.
[3] La capacidad total considera la capacidad de todos los carros de la montaña rusa.

2. `SPACE-LEFT {id_montana}`

   Indica la cantidad de asientos libres de la montaña rusa con id `id_montana`.

   output

   ```
   La montana rusa {id_montana} tiene {cantidad} asientos libres.
   ```

3. `STATUS`[4]

   Muestra ordenados[5] todos los carros de cada montaña rusa, y la cantidad de gente adentro. Imprime lo siguiente:

   output

   ```
   STATUS:
   Montana rusa 0 - {cantidad_montana_0}
       Carro {id_1}
       ...
       Carro {id_K}

   Montana rusa 1 - {cantidad_montana_1}
       Carro {id_1}
       ...
       Carro {id_K}

   Montana rusa N-1 - {cantidad_montana_N-1}
       Carro {id_1}
       ...
       Carro {id_K}
   ```

Parte 2: Administrando el Parque de Diversiones

1. `SIT {id_persona} {estatura} {peso} {id_montana}`

   Ingresa a una persona con los parámetros indicados a la montaña rusa de id `id_montana`. La persona se subirá al primer carro que llegó y tenga espacio disponible. Se imprime:

   output

   ```
   La persona {id_persona} ha ingresado al carro {id_carro}.
   ```

   En caso de que la montana rusa no tenga espacio disponible, entonces la persona no podrá ingresar[6] y se imprime:

   output

   ```
   La montana rusa {id_montana} se encuentra llena.
   La persona {id_persona} no pudo ingresar.
   ```

   Por otro lado, en caso de que la persona no cumpla con la estatura mínima para subir a la montaña rusa, entonces la persona no podrá ingresar y se imprime:

   output

   ```
   La persona {id_persona} no cumple con la estatura minima.
   ```

   Finalmente, en caso de que al subir la persona se supere el peso máximo que soporta la montaña rusa, entonces no podrá ingresar y se imprime:

[4] Sólo se muestran las montañas rusas que han sido construidas. Además, éstas siempre tendrán al menos 2 carros.
[5] Siguiendo el orden en que los carros fueron entregados en el evento BUILD.
[6] En cualquier caso en que la persona no pueda ingresar, esta simplemente se va, es decir, no queda esperando en alguna cola.

```
output
1 La persona {id_persona} no pudo ingresar.
2 Se superaria el peso maximo permitido.
```

2. `CAR {id_montana} {id_carro}`
   Muestra en orden de llegada a todas las personas a bordo del carro {id_carro} y su cantidad de pasajeros. El carro se encuentra en la montaña rusa con id {id_montana}. Imprime lo siguiente:

```
output
1 Carro {id_carro} - {cantidad_pasajeros}
2     {id_persona_1}
3     {id_persona_2}
4     ...
5     ...
```

   En caso de que el carro esté vacío, imprime:

```
output
1 El carro {id_carro} se encuentra vacio.
```

3. `GET-DOWN {id_montana} {id_carro}`
   Se baja la primera persona[7] en haber entrado al carro señalado de la montaña rusa recibida.

```
output
1 Se ha bajado la persona {id_persona}.
```

4. `SWITCH {id_persona1} {id_persona2} {id_montana}`
   Intercambia la posición de la persona con id `id_persona1` y la persona con id `id_persona2`. Ambas estarán en la misma montaña rusa, pero en un carro distinto. Este evento no imprime output.

5. `INVERSE {id_montana}`
   La montaña rusa {id_montana} se ha quedado completamente detenida en la cima y se fue hacia atrás, por lo que se invirtió el orden de sus carros y los pasajeros de cada carro[8]. El output deberá entregar el nuevo orden de la montaña:

```
output
1 Montana rusa {id_montana} invertida:
2     Carro {id_K}
3     ...
4     ...
5     Carro {id_1}
```

Consideraciones Generales

- El primer evento siempre será un BUILD
- No habrá carros con el mismo id. Tampoco habrá personas con el mismo id.
- Todos los valores numéricos entregados son enteros positivos y pueden ser contenidos dentro de un `int`.
- No se entregará una complejidad esperada, sin embargo se debe cumplir con los tiempos de ejecución mencionados en la sección de Evaluación.

[7] Siempre habrá al menos una persona en el carro indicado.
[8] Esto afecta el orden entrada en el evento SIT, el orden de bajada en el evento GET-DOWN y el output en STATUS y CAR.

Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un ejecutable de nombre `rollercoaster` que se ejecuta con el siguiente comando:

```
./rollercoaster input output
```

donde `input` será un archivo con los eventos a simular y `output` el archivo donde debes escribir los resultados.

En caso de querer correr el programa para ver los leaks de memoria utilizando valgrind deberás utilizar el siguiente comando:

```
valgrind ./rollercoaster input output
```

Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada una de estas ejecuciones que tenga un output igual al esperado. A continuación detallaremos los archivos de input y output.

Input

Primero recibirás una línea con un entero N, indicando la cantidad de máxima de montañas rusas. Luego recibirás un entero E que indicará la cantidad de eventos a recibir. Luego recibirás los eventos y, en caso del evento BUILD, recibirás K líneas más, indicando la id de cada carro.

Como ejemplo tenemos el siguiente input:

input

```
2
9
BUILD 3 10 130 550
13
4
8
SIT 4 150 60 0
SIT 8 130 45 0
BUILD 2 6 140 1000
2
10
STATUS
SPACE-LEFT 0
INVERSE 0
SIT 5 140 55 0
STATUS
```

Output

El output deberá consistir en un archivo con la información solicitada por cada evento en el archivo de input. Ojo, ten en consideración los saltos de línea, donde todas las líneas terminan con un salto. Así también, considera los espacios de cada impresión para que tu output coincida con el de los casos de prueba.

El output del ejemplo anterior es el siguiente:

output

```
Montana rusa creada, de id 0 y capacidad de 30 personas:
    Carro 13
    Carro 4
    Carro 8
La persona 4 ha ingresado al carro 13.
La persona 8 ha ingresado al carro 13.
Montana rusa creada, de id 1 y capacidad de 12 personas:
    Carro 2
    Carro 10
STATUS:
Montana rusa 0 - 2
    Carro 13
    Carro 4
    Carro 8
Montana rusa 1 - 0
    Carro 2
    Carro 10
La montana rusa 0 tiene 28 asientos libres.
Montana rusa 0 invertida:
    Carro 8
    Carro 4
    Carro 13
La persona 5 ha ingresado al carro 8.
STATUS:
Montana rusa 0 - 3
    Carro 8
    Carro 4
    Carro 13
Montana rusa 1 - 0
    Carro 2
    Carro 10
```

Evaluación

La nota de tu tarea es calculada a partir de testcases de input/output. La ponderación se descompone de la siguiente forma:

Nota entre partes (90 %)	**Manejo de memoria (10 %)**
20 % Tests Parte 1	5 % Sin leaks de memoria
70 % Tests Parte 2	5 % Sin errores de memoria

Para cada test de evaluación, tu programa deberá entregar el output correcto en **menos de 10 segundos** y utilizar menos de 1 GB de RAM[9]. De lo contrario, recibirás 0 puntos en ese test.

Recomendación de tus ayudantes

Estas tareas generalmente requieren de mucha dedicación, por lo que desde ya te recomendamos distribuir tu tiempo considerando los plazos definidos. Así mismo, te recomendamos fuertemente que antes de empezar a programar tu tarea, **leas el enunciado detenidamente y con anticipación para que te dediques a entender de manera profunda lo que te pedimos**. Una vez hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación, modelación y elaboración de tu informe, para posteriormente poder programar de manera más eficiente. No olvides compilar el código como se indica en la tarea y de preguntar cualquier duda o por problemas que te surjan en Discussions. Estos son consejos de tus ayudantes que te pueden ayudar a pasar el ramo.

Entrega

Código: GIT - Rama principal del repositorio asignado, que debes generar con el link dado al principio de este enunciado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Atraso: Esta tarea **NO** considera la política de atraso y cupones especificada en el repositorio del curso.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** por el alumno y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.

Uso de IA

Se prohíbe el uso de herramientas de inteligencia artificial para la realización de esta tarea. Esto incluye, pero no se limita a, el uso de modelos de lenguaje como ChatGPT, Copilot, u otras tecnologías similares para la generación automática de código, soluciones, o cualquier parte del trabajo requerido. Nos reservamos el derecho de revisar todas las tareas entregadas con el fin de detectar el uso de inteligencia artificial en la creación de las soluciones. Las tareas en las que se determine que se ha utilizado IA serán consideradas como una infracción a la política de honestidad académica y serán tratadas como casos de copia.

[9] Puedes revisarlo con el comando `htop` u ocupando `valgrind --tool=massif`