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

IIC2133 — Estructuras de Datos y Algoritmos
2020 - 1

Tarea 0

Fecha de entrega código: 8 de Abril
Fecha de entrega informe: 10 de Abril

Objetivos

- Familiarizar al estudiante con el lenguaje de programación C
- Implementar una variación de una estructura de datos básica
- Simular eventos secuenciales mediante las operaciones de dicha estructura de datos

Introducción

¡Suenan las alarmas a lo largo de la galaxia! Los Piratas Espaciales han tomado control del *Space Titanic*, generando una gran amenaza para **niños**, **adultos** y **robots** que se encuentran a bordo de este. La Comisión Extraterrestre de Inteligencia (CEI) debe enviar una flota de *Pods* a los distintos terminales presentes, con el fin de que la mayor cantidad de pasajeros puedan evacuar antes que los piratas tomen el total control de la nave.

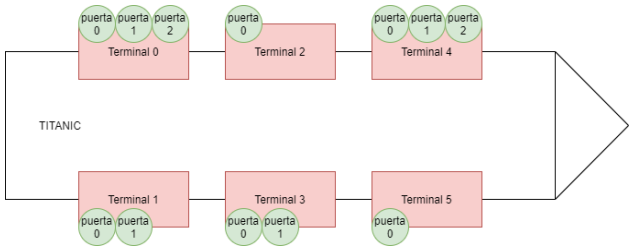
El directorio de la CEI acaba de contactar al equipo docente del curso IIC2133, para que desarrollen una simulación de la evacuación del *Space Titanic*, y así poder determinar la cantidad de *Pods* a enviar y la distribución de estos en la nave, maximizando así la cantidad de personas y robots rescatados.

Sin embargo, dado que el equipo docente se encuentra en cuarentena tras contraer la *Space Flu*, es tu labor como programador de *Space-C* coordinar el rescate!

Space Titanic

Problema

Para realizar la simulación primero hay que entender como están construidas las salidas de emergencia.
El barco tiene varias salidas de emergencia denominadas `terminales`, estas a su vez, están compuestas de varias `puertas`. Las puertas son los lugares donde llegan los pods de rescate[1].



Como los pasajeros del *Space Titanic* vienen de una utopía futurista, todos los pasajeros se forman ordenadamente en colas fuera de cada una de las puertas del barco. Cuando un *Pod* llegue a una puerta del barco, los primeros 8 pasajeros que estén haciendo cola en esa puerta se subirán al *Pod* y volverán a la tierra. Luego, todos los pasajeros en aquella cola que estén desde la posición 9 en adelante avanzarán 8 espacios dentro de su cola.
Puedes asumir que siempre habrán al menos 8 pasajeros en la cola una vez que llegue el pod, además puedes asumir que todas las colas comienzan vacías.

Orden de Evacuación

Como en todo proceso de evacuación, existe un código de conducta con respecto a la prioridad de las vidas que salvar. Dado que la CEI sigue la Ley Marítima Espacial 18520, siempre deberás evacuar a los pasajeros en el siguiente orden.

1. Niños
2. Adultos
3. Robots

Por lo tanto, para cada cola dentro del titanic, siempre deben estar los niños al inicio de ella, luego los adultos y por último los robots. Por ejemplo, en caso de llegar un nuevo niño a la cola, este se colocará detrás de todos los niños que estaban desde antes, pero delante de todos los adultos y robots.

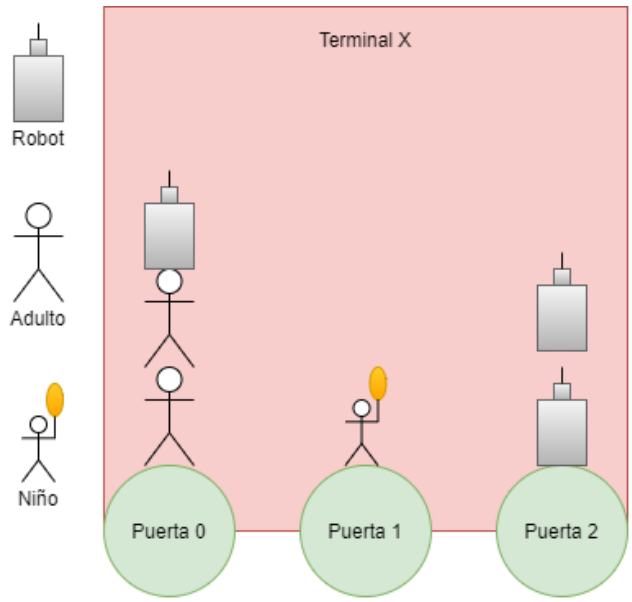
[1] https://en.wikipedia.org/wiki/Escape_pod

Eventos

Un evento representa una situación que ocurre dentro del barco, estos permitirán simular de manera exacta como funcionaría el rescate. Cada línea del archivo de *input* de su simulación contiene uno de estos eventos, los cuales se ejecutarán en el mismo orden que se encuentra en el archivo.

`INGRESO terminal(int) ID_pasajero(int) tipo(int)`

Un pasajero de id `ID_pasajero` y tipo `tipo` ((0) niño, (1) adulto o (2) robot) que se encuentra en el *Space Titanic* ingresa al terminal `terminal`. Este pasajero escoge la puerta más favorable del terminal, es decir, en donde pueda quedar en una mejor posición. Para el caso de los niños, solo observan la cantidad de niños en cada cola, mientras que adultos observan cantidad de niños y adultos, y finalmente robots observan la cantidad total de personas en la cola para esa puerta. En caso de que dos puertas sean igualmente favorables, se escoge la de menor índice en el terminal.
Por ejemplo, si tienen las siguientes 3 colas frente a las 3 puertas del terminal X:



En caso de llegar un niño al terminal, este quedaría en la posición 0 de la puerta 0 (ya que tiene prioridad sobre los adultos y robots), en la posición 1 de la puerta 1 y en la posición 0 de a puerta 2. Como queda en la misma posición en las puertas 0 y 2 el niño ingresa a la puerta 0 ya que tiene menor índice.

`ABORDAJE terminal(int) puerta(int)`

Abordan los primeros 8 pasajeros en la cola de la puerta `puerta` del terminal `terminal`. Deberás imprimir en consola: `POD terminal(int) puerta(int) indice(int) LOG`, seguido de los id en orden de todos los pasajeros que subieron al *pod*. El índice es el número del *pod* que sale desde esa puerta, comenzando desde 0. Por ejemplo, si se aborda por segunda vez un *pod* desde la puerta 5 del terminal 2. Entonces se debe imprimir en por consola:

```
POD 2 5 1 LOG
id_pasajero0
id_pasajero1
id_pasajero2
```

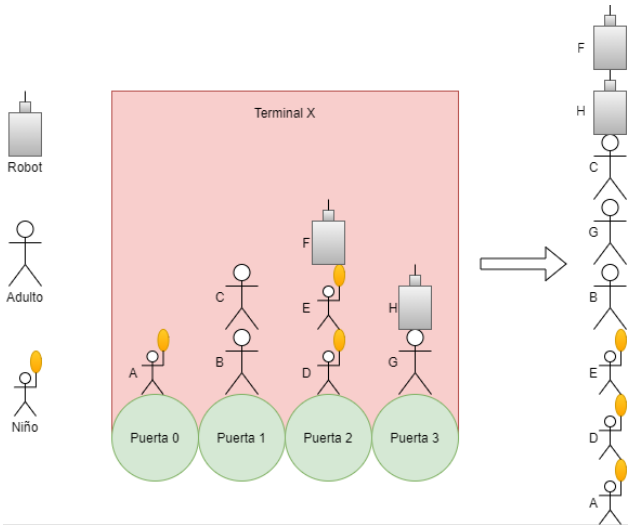
```
id_pasajero3
id_pasajero4
id_pasajero5
id_pasajero6
id_pasajero7
```

`CIERRE terminal(int) puerta(int)`

Se cierra la puerta `puerta` del terminal `terminal`. Desde ahora en adelante esta puerta no se podrá volver a abrir, por lo que se debe ignorar al momento de ingresar nuevos pasajeros al terminal. Los pasajeros que se encuentren actualmente en la cola de la puerta cerrada deberán reingresar al mismo terminal (a otras puertas).
Puedes asumir que a lo menos habrá una puerta abierta por terminal, nunca van a estar todas cerradas.

`CLAUSURA terminal1(int) terminal2(int)`

Se inhabilita el terminal `terminal1`, trasladando a toda la gente al terminal `terminal2`. Como el terminal `terminal1` puede tener varias puertas aún abiertas, todas las colas de las puertas abiertas dentro del `terminal1` se deben mezclar en una sola cola, luego, todos los pasajeros de esta nueva cola deben ser ingresados en orden a la `terminal2`.
Para mezclar estas colas se debe hacer intercaladamente. Por ejemplo, en el caso de haber 3 puertas, el primer pasajero en la nueva cola será el primer pasajero en la puerta 0, luego el primer pasajero en la puerta 1, luego el primer pasajero en la puerta 2, luego el segundo pasajero de la puerta 0 y así.
Es importante recordar que en esta nueva cola se debe mantener el orden de evacuación (primero niños, luego adultos y por último robots).
La siguiente figura muestra un ejemplo de como quedaría la nueva fila formada por la clausura del terminal X.



`LASER terminal(int) puerta(int) índice(int)`

Un rayo láser descontrolado lanzado por los Piratas Espaciales provoca la muerte de la persona en el lugar `índice` de la cola existente en la puerta `puerta` del terminal `terminal`. Todos los pasajeros de la cola que estaban detras del pasajero atacado avanzan una posición en la cola.

Input

Para correr tu simulación esta se ejecutará junto a un archivo `test.txt`, el cual contiene toda la información del Titanic y los eventos que ocurren. La información en `test.txt` viene entregada de la siguiente forma:

- La primera línea tiene un unico número N que indica la cantidad de terminales
- Luego, N lineas, donde la iésima linea indica la cantidad P_i de puertas que tiene el terminal i
- Luego, cada línea contiene un evento, como fueron detallados en la sección Eventos
- Finalmente, una línea conteniendo la palabra `END` que indica el fin del archivo

Un ejemplo del archivo input es el siguiente:

```
2
2
8
INGRESO 0 0 1
INGRESO 0 3 1
INGRESO 0 23 0
INGRESO 0 7 2
INGRESO 0 5 1
INGRESO 0 64 1
INGRESO 0 31 2
LASER 0 1 1
INGRESO 0 5 0
INGRESO 0 6 1
INGRESO 0 91 1
CIERRE 0 1
ABORDAJE 0 0
INGRESO 0 129 2
END
```

Puedes asumir que los eventos detallados en el archivo de input siempre serán válidos y ejecutables para el estado que debería tener el programa. En particular:

- Si una puerta inicia abordaje, entonces hay al menos 8 personas haciendo fila en esa puerta
- El índice del láser siempre estará dentro de la fila.
- Siempre habrá al menos una puerta abierta por terminal, y el abordaje siempre será en una puerta abierta.
- Siempre habrá al menos un terminal abierto, y el ingreso siempre será en un terminal abierto.

Output

Además de los `printf` cada vez que se aborda un *pod*, también deberás imprimir todos los pasajeros que quedan en el Titanic al momento de finalizar el programa, estos prints son conocidos como el `TITANIC LOG`. El formato de estos `printf` es el siguiente: para cada terminal no clausurado y para cada puerta no cerrada dentro de este terminal debes imprimir los ID de todos los pasajeros haciendo fila en esa puerta, indicando cuantos son. Por ejemplo:

```
TITANIC LOG
TERMINAL 0
GATE 0: A
```

```
id_pasajero0
...
id_pasajeroA-1
GATE 1: B
id_pasajero0
...
id_pasajeroB-1
...
TERMINAL N
GATE 0: C
id_pasajero0
...
id_pasajeroC-1
...
END LOG
```

En los casos de terminales clausurados o puertas cerradas estos no se deben imprimir y los debes saltar en el `TITANIC LOG`.
Recomendamos ver los ejemplos subidos para que tus prints finales sean los iguales a los esperados.

Tu programa se debe compilar con el comando `make` y debe generar un ejecutable de nombre `simulate` que se ejecuta con `./simulate test.txt` donde `test.txt` es el archivo de input que se le pasa al programa.
Para corregir tu tarea se evaluará ejecutándola con la siguiente línea: `./simulate test.txt > out.txt`
Al correr esa línea se creará el archivo `out.txt` en la carpeta donde se ejecute, la cual contendrá todos los prints del archivo `simulate`.

Análisis

Deberás escribir un informe de análisis donde menciones los siguientes puntos:

- Calcula y justifica la complejidad en notación $\mathcal{O}$ para cada uno de los eventos en función de la cantidad de personas, la cantidad de puertas y la cantidad de terminales. Debes simplificar la expresión lo más posible.

Uso de memoria

Parte de los objetivos de esta tarea es que trabajen pidiendo y liberando memoria. Para evaluar esto usaremos una herramienta llamada *valgrind*, la cual permite revisar si tienen errores de memoria y problemas de *memory leaks*. Se recomienda leer el código del Taller 0 - Intro a C. El archivo `src/6_memoria/main.c` incluye en sus comentarios como interpretar el output de *valgrind*.

Evaluación

La nota de tu tarea se descompone como se detalla a continuación:

- 70% a la nota de tu código, dividido en:
  - 25% que el output de tu programa sea correcto.
  - 25% que tu programa sea eficiente. Para esto, tu código será ejecutado con inputs grandes y deberá correr en menos de 10 segundos por input.

  - 10% que *valgrind* diga que tu programa no tiene errores de memoria.
  - 10% que *valgrind* diga que tu programa no tiene *memory leaks*.
- 30% a la nota del informe, basada en el cálculo de la complejidad teórica y justificación para cada uno de los eventos.

Entrega

Código: GIT - Repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Informe: SIDING - En el cuestionario correspondiente, en formato PDF. Sigue las instrucciones del cuestionario. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.