Parte A: Resolución de problemas algorítmicos

Integrantes de la resolución:	***Borquez, Juan Manuel***	***13567***
	Dalessandro, Francisco	***13318***
	Panonto, Valentín	***13583***
	Welch, Patricio	***13612***
	Zamora, Braulio	***13571***

Para cada uno de los siguientes enunciados:

a. **Realizar una lectura comprensiva de los textos en grupos de ~~no más de 4~~ personas y discútelo en grupo**. Nótese que los diferentes ejercicios tienen distintos niveles de detalle. Esto tiene como objetivo que en el análisis general del problema puedan identificar información faltante (si la hubiera).

b. Identificar: entradas, salidas, información relevante, requerimientos y restricciones.

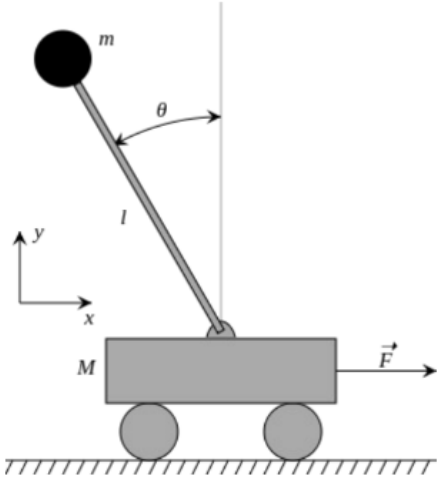
c. Elaborar un algoritmo general de **alto nivel de abstracción** que resuelva el problema.

d. Una vez realizado el algoritmo de alto nivel, identifique al menos 3 funciones que implementarían alguno de los pasos del algoritmo. Indique la forma completa de la función (nombre, valor de retorno, parámetros con sus tipos y nombres de variable) y una breve documentación de la misma.

EJERCICIO 1

Un problema clásico de la literatura de Control consiste en mantener en posición vertical un péndulo invertido fijado en un eje a un carrito que puede moverse a izquierda y derecha (ver imagen). La señal de control indica la fuerza "*F*" a aplicar sobre el carrito para balancear el péndulo. El carrito tiene una masa "*M*", el péndulo tiene una masa "*m*" y una longitud "*L*", existe un límite a la izquierda y derecha, más allá del cual el carrito no puede moverse.

Se desea desarrollar un simulador del carrito y un controlador para el mismo.



a) **Lectura comprensiva**
b) **Identificación del problema**

Variables de ENTRADA:
- ”M” (masa del carrito).
- ”m” (masa del péndulo).
- ”L” (longitud de la barra).
- “θ” (ángulo que forma la barra con la vertical).

Variables de SALIDA:
- ”F” (Fuerza a aplicar sobre el carro para balancear el péndulo).

Información relevante:
- Posición del carrito con respecto a su origen y/o a sus límites
- Posición de los límites

Requerimientos:
- Mantener en posición vertical el péndulo invertido.

Restricciones:
- Existe un límite a la izquierda y derecha, más allá del cual el carrito no puede moverse.

c) **Algoritmo de alto nivel de abstracción**

Inicio del programa
Declarar y Definir las variables y constantes de entrada.
Realizar lectura del ángulo de inclinación "θ" de la masa “m”.
Calcular la fuerza necesaria para regresar la masa al punto central con ángulo "θ=0°”.
Enviar como salida la Fuerza a aplicar.
Fin del programa.

d) **Funciones a implementar**

Primera función:

```c
int main(void) {
  float m, M, L, tita;
  printf("Ingrese masa del péndulo, masa del carrito, longitud de la vara y
ángulo inicial");
  scanf("%f %f %f %f", &m, &M, &L, &tita);
```

Segunda función:

```c
float calculo_fuerza (float m, float M, float L, float tita) {
  float F, p, pmin, pmax; //Fuerza, posición, posición mínima, posición
máxima
  //Cálculo de la fuerza de salida
  if (p == pmin || p == pmax) {
    printf("El carrito se pasó de los límites y no se puede equilibrar");
  } else {
    return F;
  }
}
```

Tercera función:

```
void imprimirResultados(float F){
  printf("La fuerza aplicada para mantener el equilibrio es %f", F);
}
```

EJERCICIO 2

La “Batalla naval espacial” se juega en un tablero de 4 x 4, donde las filas se identifican de la A hasta la D y las columnas del 1 al 4. En el juego participan 2 contendientes: el defensor y el atacante. Dicho juego consiste en:

El *defensor*, ubica solo una nave nodriza triple con ciertas reglas:

• La nave debe ubicarse de tal forma que sus partes queden contiguas, ya sea horizontal o vertical, pero no es válido en forma oblicua.

• Cada una de las tres partes que componen la nave contiene un escudo de electrones medido con un valor del 1 al 9, el cual debe pedirse al usuario junto con su posición.

A continuación, se ilustra un ejemplo de una ubicación posible:

	1	2	3	4
A				
B				
C		4	7	1
D				

El *atacante*, indicando una coordenada del tablero (por ejemplo, C3) y una carga de protones, debe intentar acertar a la nave de su contrincante. El ataque, posee las siguientes reglas:

1. La carga de protones asociada al ataque corresponde a un valor del 1 al 9.
2. Si el atacante no acierta en la posición, entonces el defensor informa “Espacio!”.
3. Si el atacante acierta la posición,
    1. El ataque es “efectivo” y resta el valor de la carga protones al escudo de electrones, si y sólo sí, el valor de la carga de protones es menor o igual al valor restante de electrones del escudo. En el ejemplo de ubicación anterior si el atacante indica C3 con carga 9, el ataque es “sin efecto” y no genera daño alguno. Pero si indica C3 con carga 4 el ataque es “efectivo” y el escudo de la posición queda con carga de 3 electrones.
    2. Luego del ataque se debe indicar si fue efectivo o no, si se neutralizó o no el escudo del casillero y la suma total de electrones que resta para hundir la

nave. El escudo de un casillero se neutraliza cuando llega a cero. Suponiendo que en el primer ataque se indica C3 con carga 4, se indica “Ataque efectivo – Escudo no neutralizado – Carga restante de electrones igual a 3”.

4. Cada vez que el atacante realiza un disparo resta el valor de la carga de su reactor de protones. El reactor de la nave atacante es de 40 protones. Un disparo a realizar no puede superar la carga de protones restantes.

El juego termina cuando se cumple alguna de las siguientes situaciones:

• Gana el atacante cuando deja sin escudos a la nave nodriza y todavía le queda carga para un disparo más.

• Gana el defensor cuando el atacante se queda sin carga en el reactor de protones.

a) **Lectura comprensiva**
b) **Identificación del problema**

Variables de ENTRADA:

- Coordenadas de la casilla a disparar.
- Cantidad de protones a disparar en cada turno.

Variables de SALIDA:

- Disparo efectivo o no.
- Cantidad de electrones restantes de la posición (si es que tenía algo).
- Cantidad de protones restantes.
- Escudo neutralizado o no.
- Victoria o derrota.

Información relevante:

- Las posiciones de los escudos del defensor y la cantidad de electrones de los mismos.

Requerimientos:

- Generar una matriz de 4x4 y en la misma asociar a 3 posiciones consecutivas en horizontal o vertical (no en diagonal) una cantidad aleatoria de electrones del 1 y al 9.
- Pedir al atacante las coordenadas y la cantidad de protones de cada disparo.
- Informar si un ataque fue efectivo o no y la cantidad de electrones restantes del escudo en el caso de ataque efectivo.
- Informar la cantidad de protones restantes.
- Informar si se obtuvo victoria o si fue una derrota.

Restricciones:

- La cantidad de protones a disparar debe ser mayor o igual a 1 y menor o igual a 9.
- No se puede ingresar una cantidad de protones mayor a la que tienes como jugador.
- Los escudos generados son casillas contiguas en vertical u horizontal.
- La matriz ha de ser 4x4.
- No se puede ingresar una coordenada fuera de los límites de la casilla.

c) **Algoritmo de alto nivel de abstracción**

- Crear la matriz de 4x4.- Generar los escudos con una cantidad aleatoria de electrones del 1 al 9 en
tres posiciones consecutivas en vertical u horizontal.
- Pedir al usuario las coordenadas del disparo.
- Verificar que las coordenadas indicadas sean correctas, sino pedir una nueva coordenada en caso de que sea incorrecta hasta que sea correcta.
- Pedir la cantidad de protones del disparo. Verificar que la cantidad de protones sea correcta (un número del 1 al 9), sino pedir una nueva cantidad hasta que sea correcta.
- Comparar la coordenada del disparo con las coordenadas ocupadas por los escudos.
- Mostrar "espacio" en caso de que la coordenada indicada no esté ocupada y en caso de estar ocupada comparar la cantidad de protones del disparo con la cantidad de electrones de la casilla. Si la cantidad de protones es mayor a la cantidad de electrones mostrar "Ataque sin efecto". En caso contrario restar a la cantidad de electrones de la casilla la cantidad de protones del disparo. Mostrar "Ataque efectivo-escudo neutralizado" si la cantidad de electrones restantes en la casilla es cero y mostrar "Ataque efectivo-escudo no neutralizado-electrones restantes" en otro caso.
- Restar los protones utilizados al total de protones restantes.
- De acuerdo a si el jugador o se quedó sin protones o el defensor sin escudos (lo que pase primero), es que se determina si el jugador gana o pierde: pierde en el primer caso y gana en el segundo. En cualquiera de los casos el juego se termina.
- Si no se dio ninguna de las condiciones anteriores volver a pedir un ataque al atacante.

d) **Funciones a implementar**

Primera función:

```
int** matriz_gen(int *i1, int *j1, int *i2, int *j2, int *i3, int *j3, int *e1, int *e2, int *e3){
  //*i1, *j1 son los indices en la matriz del primer escudo
  //*i2, *j2 son los indices en la matriz del segundo escudo
  //*i3, *j3 son los indices en la matriz del tercer escudo
  //*e1 es la cantidad de electrones del primer escudo
  //*e2 es la cantidad de electrones del segundo escudo
  //*e3 es la cantidad de electrones del tercer escudo

  /*Genera la matriz de 4x4 y en tres posiciones consecutivas aleatorias de la misma (horizontal
o vertical) coloca numeros aleatorios del 1 al 9 */
}
```

Segunda función:

```
void protones(int P, int p){
  //Resta la cantidad de protones usados a la cantidad de protones restante
}
```

Tercera función:

```
void insertarAtaque(int coor1, int coor2, int coor3){
    //Registra la casilla a atacar y la cantidad de protones a utilizar
}
```

Cuarta función:

```
int comparador (int coor1, int coor2, int coor3){
    //El comparador arrojará un 1 cuando el ataque sea correcto y un 0 cuando sea incorrecto
}
```

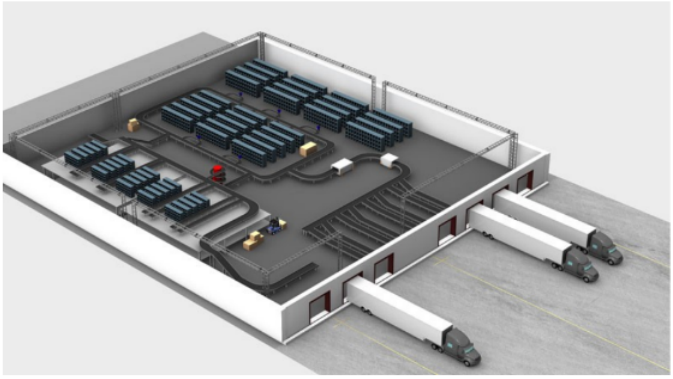
Quinta función:

```
int terminarJuego(int cant_prot, int escudos){
    //Devuelve 1 si la cantidad de protones total es 0, o si los escudos ya no tienen electrones y devuelve 0 si no pasa ninguna de las dos
}
```

EJERCICIO 3

En un Centro de Distribución (CD), los productos se encuentran almacenados en posiciones que se identifican por su pasillo, estantería y nivel. Las Órdenes de Pedido (OP) que llegan al CD, asociadas a un determinado cliente (con su línea de crédito correspondiente, dirección y contacto), contienen una lista de productos y cantidades que deben ser cargadas en un camión y despachadas. A cada OP se le asigna un camión, el cual se ubica en una bahía de carga determinada (el CD tiene varias bahías de carga para cargar camiones en paralelo). El "picking" (la tarea de tomar uno o más productos de las estanterías y colocarlos en la bahía de carga correspondiente) es realizado por el personal del CD (llamados "pickers") mediante distintos tipos de vehículo para trasladar pallets. A cada picker se le actualiza automáticamente la lista de productos (incluyendo pasillo, estantería y nivel de donde debe tomarlos, y su cantidad) mediante un enlace de radio-frecuencia, y se le indica la bahía de carga donde debe dejarlos.
Los camiones tienen capacidad limitada, cuando se llenan, se despachan.

Se debe desarrollar el sistema que planifica las operaciones de picking del CD.



a) **<u>Lectura comprensiva</u>**

b) Identificación del problema

Variables de ENTRADA:

- Datos del cliente (línea de crédito correspondiente, dirección y contacto)
- Orden del pedido "OP" (se incluye cada producto y cantidad, con el respectivo: pasillo, estantería y nivel en el que se encuentra).

Variables de SALIDA:

- Camión asignado.

Información relevante:

- Datos del cliente, para verificar que esté en la lista de clientes.
- Datos del producto, para verificar que estén en la lista propia del centro.
- Capacidad máxima del camión.

Requerimientos:

- Asignar los productos correctamente para llenar el camión y que sea despachado.

Restricciones:

- Los camiones tienen capacidad limitada.

c) Algoritmo de alto nivel de abstracción

Inicio del programa

Declarar y Definir las variables de entrada.

Realizar lectura de los datos referidos al cliente y verificar que el mismo se encuentre en la lista de clientes.

Realizar lectura de los datos propios de la OP y verificar que los mismos se encuentren en la lista de productos.

Ubicar y asignar la carga al camión correspondiente según la bahía en la que se encuentra.

Enviar como salida la bahía del camión asignado.

Fin del programa.

d) Funciones a implementar

Primera función:

```
//Primera función: Verificación de que el cliente esté en la lista.
int cliente(struct cliente* lista_clientes){
  /*Recibe la lista de clientes*/
  while (/*mientras el cliente no haga coincidencia con alguno de la lista*/){
    /*Se desplaza en una posición a la lista de clientes*/
  }
  if(/*Si el cliente se encontró en la lista*/){
      /*Cliente válido, se pasa a analizar el pedido*/
    }else{
      /*Cliente inválido, se descarta su pedido*/
  }
  return valid_cliente;
}
```

Segunda función:

```c
//Segunda función: Verificación de que el pedido sea válido.
int verificacion(struct productos* lista_productos, struct pedidos* lista_pedido){
  /*Recibe la lista de los productos en el almacén*/
  /*Recibe la lista de los productos pedidos*/
  /*Ingresa al siguiente while con el producto y lo compara con todos los del almacén*/

  while(/*La lista de productos pedidos por el cliente no se haya acabado*/){
    bandera=0;
    while(bandera == 0 && /*la lista de productos del almacén no se haya acabado*/){
      /*Compara el producto que el cliente pide con el producto que hay en el almacén*/
      if(/*Si encuentra coincidencia entre el pedido y el producto del almacén*/){
        /*Cambia al siguiente producto pedido por el cliente*/
        bandera=1;
      }
      /*Se desplaza en la lista de los productos del almacén*/
      if(/*Si la lista de productos del almacén se recorrió completamente y no se encontró coincidencia entre el producto que se pide y los que hay en el almacén (ejemplo de si pedimos un lavarropas a una central de bebidas)*/){
        /*El pedido debe ser descartado*/
      }
    }
    /*Se desplaza en la lista de los productos pedidos por el cliente*/
  }
}
```

Tercera función:

```c
//Tercera función: Asignación y verificación del camión.
void carga_camion(struct camiones* lista_camiones, struct pedidos* lista_pedido){
  /*camiones es una lista la que cada nodo referencia a un camión. Contiene el número del camión(igual al nro de bahía en el que se encuentra ese camión) y el nro de productos restantes para que tenga carga completa*/
  char camion_asignado[20];  //Si se asigna carga al camión 1 y 2, se debe escribir '1-2' en esta variable
  /*La lista del pedido, también incluye el número de artículos que se han pedido y un camión asignado con valor incial 0*/

  /*Se asigna el camión correspondiente a la lista del pedido*/
  /*Se actualiza el valor del camión asignado al número o a los números de camión/es asignado/os */
  /*Si la carga excede a la capacidad de un solo camión, se debe asignar parte al siguiente camión en la lista de camiones*/

  /*Imprimimos los resultados de aquellos camiones que están listos para ser despachados*/
  while(/*La lista de camiones no se haya terminado*/){
    if(/*Si el camión asignado se encuentra con carga completa*/){
      printf("El camión de la bahía %d ha sido despachado con carga completa", /*apuntador al nro correspondiente*/);
    }else{
      printf("Ningún camión ha sido llenado con la carga");
      }
    /*se pasa al siguiente nodo de la lista*/
  }
}
```

EJERCICIO 4

El “Tren Subacuático” es la atracción principal del “Cataratas Park” y tiene salidas programadas cada 15 minutos. Este tren posee dos vagones de seis asientos cada uno. Cuando un grupo de visitantes llega a la atracción, informa la cantidad de pasajeros que compone el grupo. El sistema de turnos genera un ID (número aleatorio de cuatro dígitos) que se utiliza para identificar al grupo y realiza la asignación de los lugares en el tren de acuerdo a la siguiente lógica:

1. En primer lugar el sistema busca si en algún vagón hay lugar suficiente para alojar a todos los pasajeros juntos, y de ser así asigna los lugares. Por ejemplo: si llega un grupo de cuatro personas y el primer vagón tiene sólo dos asientos libres pero el segundo vagón tiene cinco, entonces el sistema asigna a los pasajeros cuatro lugares del segundo vagón.
2. Cuando el sistema no encuentra lugar para que estén todos los visitantes de un grupo juntos en un mismo vagón, busca si hay lugar disponible entre los dos vagones, y si lo hay, entonces asigna los lugares comenzando por el primer vagón. Por ejemplo: si llega un grupo de cuatro personas y hay sólo dos lugares libres en el primer vagón y tres en el segundo, asigna dos asientos del primer vagón y dos asientos del segundo.
3. En el caso en que no haya lugar para alojar a todo el grupo en el tren tendrán que esperar la próxima salida, lo cual es informado al usuario.

El sistema sólo permite salir al tren cuando cumple algunas de estas condiciones:

1. Cuando el tren está lleno.
2. Cuando no hay más grupos para cargar, es decir que no hay más gente en la fila.
3. Cuando el último grupo que se intentó cargar no entró en el tren (lo explicado en el punto 3 anterior). NOTA: es importante considerar que un tren ***nunca*** puede salir vacío.

Realice un programa que permita realizar la carga de los grupos de visitantes, asigne los asientos, y que una vez que detecte que el tren puede salir, emita un listado de cómo se asignaron los lugares en el tren para cada grupo con su ID y los asientos libres.

a) **Lectura comprensiva**
b) **Identificación del problema**

Variables de ENTRADA:

- Cantidad de pasajeros que forman el grupo

Variables de SALIDA:

- Tren lleno o imposible de llenar, pero listo para salir
- Vagón y tren asignado a cada grupo

Información relevante:

- El tren no puede salir vacío.
- El tren sale si no hay más gente en la fila.
- Se debe asignar el ID a cada grupo
- Se pueden separar grupos en distintos vagones, pero no en distintos trenes.

- Si hay espacio en el tren, pero el primer grupo en la fila no entra y alguno de los grupos siguientes sí, se priorizará el primer grupo en la fila que quepa en los asientos libres, en virtud de llenar el tren y que se reduzca la probabilidad de que salga vacío.

Requerimientos:

- Asignar los grupos a vagones para que las personas viajen.

Restricciones:

- Los grupos no pueden ser de más de 12 personas (debido a que un grupo no puede estar en más de un tren y la capacidad máxima del tren es de 12 personas en sus dos vagones).
- El tren nunca puede salir vacío.

c) Algoritmo de alto nivel de abstracción

Inicio del programa

Primero se lee la cantidad de personas que pertenecen a cada grupo en la fila y se les asigna un ID que los identificará dentro del sistema.

Comenzando desde el principio de la fila se van asignando las personas de cada grupo a un vagón y tren correspondiente priorizando el llenado por completo de los mismos.

Se informa el vagón y tren que le corresponde a cada grupo.

Se informa la cantidad de ocupantes que hay en cada tren que fue asignado por el sistema.

Fin del programa.

d) Funciones a implementar

Primera función:

```
//Primera función: asignación del ID al grupo.
int ID(struct nodo_grupo* grupos){
  /*Recibe la lista de grupos*/
  /*Genera un numero de ID aleatorio de 4 digitos enteros del 0000 al 9999*/
  while (/*mientras el ID generado sea distinto a los otros de la lista*/){
    /*Compara que el número de ID no sea igual a alguno de otro grupo*/
    if(/*Si el ID coincide con alguno*/){
        /*Se debe generar uno nuevo y volver a ingresar a la estructura*/
    }else{
      /*Sale efectivo y se concidera nula la coincidencia*/
  }
}
```

Segunda función:

```
//Segunda función: chequeo asignación de cada grupo a un tren y vagón.
void asignar(struct nodo_grupo* G, struct nodo_tren* trenes){
  /*Recibe:
      1 lista trenes: es la lista con los trenes
      1 puntero G a un nodo de la lista de grupos G */

/* La función recorre la lista de trenes en busca de un tren en el que el grupo del nodo G
pueda entrar y actualiza:
      - el tren que ocupa el grupo
      - el vagón que ocupa el grupo
      - la lista de trenes*/
}
```

Tercera función:

```
//Tercera función: verificación de la capacidad ocupada de cada tren que fue
asignado.
unsigned short int(struct tren T){
  /*recibe una estructura tipo tren T que tiene como atributos: El número de tren
(tipo int) y la cantidad de personas en cada vagón del tren (dos variables tipo
int)*/
  /*Devuelve un 1 si el tren esta lleno o un 0 si quedan posiciónes vacias en el
tren*/
  return full_train;
}
```