INF-253 Lenguajes de Programación
Tarea 2: C

Profesor: José Luis Martí Lara
Ayudante de Cátedras: Sebastián Godínez San Martín
Ayudantes de Tareas: Sebastián Campos Muñoz - Gabriel Carmona Tabja

4 de mayo de 2020

1. El accidente de Mario

Mario sufrió un terrible accidente mientras tenía una de sus aventuras. Resulta que en su camino cotidiano para ir a salvar a la princesa Peach, no logró levantar su puño para golpear un bloque de concreto pegándole con la cabeza, sufriendo una contusión encéfalo craneal dejándolo en coma por varios meses. Al despertar, los doctores se llevaron la horrible sorpresa de que Mario había olvidado como ser él mismo. Su hermano Guiseppe, al recibir la noticia, se puso muy triste y pensó en hacer un programa en C que lo ayudará a recuperar la memoria, pero como está demasiado ocupado contando sus millones gracias a su exitosisimo lenguaje de programación Yahoo, te delegó la tarea a ti.

2. Escenario

En el mundo existen cinco tipos de elementos, con sus respectivas características:

1. Personaje:
   - string: nombre
   - int: estado, 1: normal, 2: brillante, 3: mareado, 4: wario
   - int: cantidad de monedas
2. Objeto:
   - string: nombre
   - int: estado que otorga, 1: health, 2: mareado, 3: wario
3. Enemigo:
   - string: nombre
4. Bloque:
   - void: contenido del bloque
   - int: tipo que almacena, 0: monedas (int) o 1: Objeto

5. Suelo:
    - int: 0: piso firme o 1: agujero infinito

Existirá una lista enlazada donde cada nodo podrá almacenera alguno de estos cinco elementos, siguiendo la siguiente estructura:

```
typedef struct escenario{
    casilla* curr;
    casilla* head;
    casilla* tail;
    int length;
    int posCurr;
} escenario;

typedef struct elemento{
    void* caracteristica1;
    void* caracteristica2;
    void* caracteristica3;
} elemento;

typedef struct casilla{
    elemento cosa;
    int tipo;
    casilla* next;
} casilla;
```

Se puede ver como cada casilla del escenario tiene un int para determinar que cosa hay en esa casilla, estos corresponden a los indicados anteriormente. Ahora, cada elemento es una estructura que consiste de 3 características máximo, puede tener menos.

Además, el escenario implementará las siguientes funciones:

1. MOSTRAR

   Muestra el contenido de la casilla i, se tiene que mostrar de la siguiente forma:

   ```
   Tipo
   Nombre_Caracteristica 1: valor
   Nombre_Caracteristica 2: valor
   Nombre_Caracteristica 3: valor
   ```

   Ejemplo, si quiero mostrar una casilla que contiene un Personaje llamado Baby Mario que esta en estado mareado y con 0 monedas.

   ```
   Personaje
   nombre: Baby Mario
   estado: Mareado
   Cantidad de monedas: 0
   ```

2. MOSTRAR_TODO

   Mostrar el contenido de cada casilla como se muestra en la función mostrar yendo de la posición 0 hasta el final.

3. ANADIR_ELEMENTO

   Añade un elemento en la posición i.

4. BORRAR_ELEMENTO

   Borra un elemento en la posición i.

5. BORRAR_TODO

   Borra todo el escenario (no mario no D:)

3. Acciones

El personaje tendrá una serie de acciones a su disposición, las cuales son:

1. SALTAR_ENCIMA

   Saltar encima recibirá el escenario y un entero, el entero señalará cuanto espacio debe saltar en relación a la posición actual del personaje (Ej: -1 salta uno hacia atrás, 2 salta dos hacia adelante)

2. CAMINAR_EN

   Caminar_en recibirá el escenario y deberá mover el personaje a la siguiente casilla disponible.

3. TACLEAR_AL_FRENTE

   Taclear al frente recibirá el escenario y deberá mover el personaje una casilla hacia la derecha.

4. RUSHEAR

   Rushear recibirá el escenario y moverá al personaje tres casillas a la derecha.

5. VER_PLATA

   Ver plata muestra la cantidad de monedas que lleva recolectada el personaje por el momento.

Cada acción corresponderá a una función. Las acciones no siempre se podrán realizar, ya que dependerán del estado del personaje. Para saber que acciones están disponibles, existirá una lista con puntero a funciones que contendrán todas las acciones que el personaje puede realizar con su estado actual (esto implica que las funciones que no puede realizar no deberían estar en la lista). Para eso, se propone la siguiente estructura de trabajo:

```
typedef struct acciones{
    dun* curr;
    dun* head;
    dun* tail;
    int length;
    int posCurr;
} acciones;

typedef struct dun{
    void (*funcion)();
    int identificador;
}dun
```

De ésta forma, existirá una lista con el puntero Void que apuntará a la función y un identificador que indicará que función es.

4. Toda acción tiene una consecuencia

Cada acción a realizar tiene sus consecuencias y requisitos, a continuación se listarán todos estos casos:

- Si el personaje salta encima de un enemigo, elimina al enemigo y toma su posición.
- Si salta encima de un objeto, toma su posición eliminando el objeto y ganando el estado que otorga.
- Si salta encima de un bloque, elimina el bloque y obtiene las monedas/objeto.
- Si taclea un enemigo, lo mata y toma su posición.
- Si taclea y no hay un enemigo, se queda en su misma posición y entra en el estado mareado, perdiendo la capacidad de saltar.
- Rushear sólo se puede utilizar si el personaje está en estado brillante, después de usarlo pierde el estado.
- Si Rushea, mata a todo enemigo que encuentre entre su posición actual y la posición final.
- Cualquier movimiento que se tope con un agujero, mata al personaje y lo devuelve al inicio.
- Si camina y se encuentra con un enemigo, el personaje muere y vuelve al inicio.
- Si está en estado wario y rompe un bloque con monedas, recibe el doble de monedas y pierde el estado
- Si el personaje muere, regresa al estado normal.
- El estado normal es el estado por defecto.
- Siempre se añade el personaje en la casilla inicial.
- Si el personaje toma un objeto con el estado health, el personaje vuelve al estado normal.

5. Funcionamiento

El programa deberá primero permitir crear el nivel a través de un menú por consola. Éste menú pedirá primero el tamaño del nivel (todas las casillas se inicializaran cómo piso normal) y nos permitirá usar las funciones del escenario. Una vez terminemos de crear el escenario, deberá haber una opción que nos permita pasar a la siguiente etapa. Se asume que siempre la primera casilla contendrá al personaje al inicio del juego.

En la siguiente etapa existirá un menú donde podremos nombrar a nuestro personaje, utilizar las funciones de escenario mostrar, mostrar todo y las acciones disponibles del personaje. Al momento de ingresar una acción se debe utilizar una función llamada choice, que recibirá la lista de acciones disponibles y la acción pedida.

6. Definición de funciones a realizar

```
void mostrar(escenario* esc, int i)
void mostrar_todo(escenario* esc)
void anadir_elemento(escenario* esc, elemento* ele, int i)
void borrar_elemento(escenario* esc, elemento* ele, int i)
void borrar_todo(escenario* esc)
void saltar_encima(escenario* esc, int i)
void caminar_en(escenario* esc)
void rushear(escenario* esc)
void ver_plata(escenario* esc)
void choice(int accion, acciones* listaAcc, escenario* esc)
```

7. Datos de Vital Importancia

- Pueden crear todas las funciones que quieran, pero deben implementar las funciones definidas y los struct.
- Pueden crear archivos nuevos si los necesitan.
- Consultas vía la plataforma oficial #AULA
- Si la acción que se desea realizar no está en la lista de acciones disponibles, imprima "OH OH!", si la acción se puede realizar, se debe mostar "HICE nombre_acción"
- Si el personaje quiere avanzar al estar en el final de la lista, se termina todo y debe aparecer "Gracias alumnos de lp por ayudarme a recuperar la memoria, pero tu 100 esta en otra tarea".
- Ningún String tendrá más de 100 carácteres.

8. Archivos a Entregar

Resumiendo lo anterior, los archivos mínimos a entregar son:

- README.txt
- escenario.c
- escenario.h
- acciones.c
- acciones.h
- main.c
- MAKEFILE

El archivo escenario.c debe contener las funciones implementadas relacionadas al escenario. El archivo acciones.c debe contener las funciones relacionadas a las acciones del personaje.

9. Sobre Entrega

- El código debe venir **indentado y ordenado**.
- Las funciones deberán ir comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y los que devuelve (en caso de que devuelva algo).
- Debe estar presente el archivo MAKEFILE para que se efectúe la revisión, este debe compilar **TODOS** los archivos.
- Se debe trabajar de forma individual.
- **La entrega debe realizarse en tar.gz y debe llevar el nombre: Tarea2LP_RolIntegrante.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la compilación y utilización de su programa.**
- La entrega será vía aula y el plazo máximo de entrega es hasta el **11 de Mayo a las 23:55 hora aula**.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Si el makefile no está bien realizado, la tarea no se revisará.
- Se utilizará Valgrind para el detectar los leak de memoria.

10. Calificación

Todos comienzan con nota máxima y se les irá descontando por cada punto omitido (el puntaje que aparece al lado es el máximo de puntos que se les descontara en caso de incumplimiento):

- Acciones (40 puntos (8 puntos por acción))
- Escenario (40 puntos)
  - Mostrar (10 puntos)
  - Mostrar todo (10 puntos)
  - Anadir elemento (10 puntos)
  - Borrar elemento(5 puntos)
  - Borrar todo(5 puntos)
- Jugar (20 puntos)
- Descuentos
  - Código no ordenado (-10 puntos)
  - Código no compila (-100 puntos)
  - Warning (c/u -5 puntos)
  - Falta de comentarios (-30 puntos MAX)
  - Por cada día de atraso se descontarán 20 puntos (La primera hora de atraso serán solo 10 puntos).
  - Porcentaje de leak de memoria ((1-5) % -5 puntos (6- 50 %) -15 puntos (51-100 %) -25 puntos)

En caso de existir nota negativa esta será reemplazada por un 0.