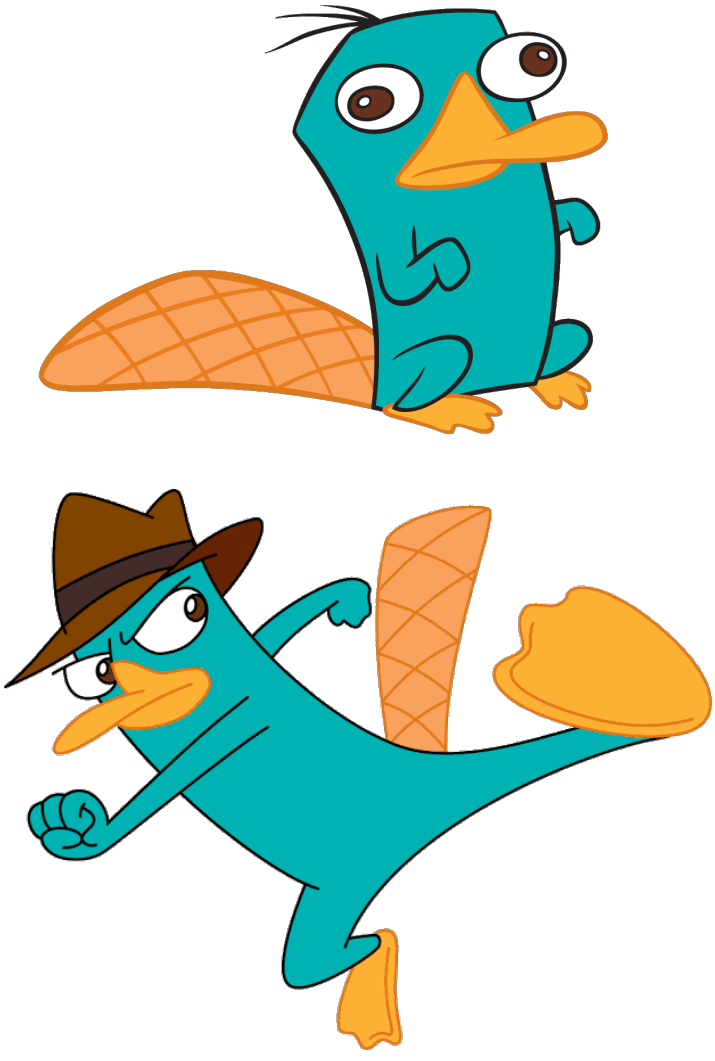
.UBAfiuba
FACULTAD DE INGENIERÍA

FUNDAMENTOS DE PROGRAMACIÓN - CURSO 4 MENDEZ

Trabajo Práctico Nº 1

¿Un ornitorrinco...? ¡Perry El Ornitorrinco!

(Primera misión)



Fecha presentación	11/04/2024
Fecha entrega	02/05/2024

Primer cuatrimestre 2024

1. Introducción

Un día soleado, el Dr. Heinz Doofenshmirtz, famoso por sus inventos y su deseo de destruir lo que le molesta debido a sus traumas infantiles, descubrió que se iba a hacer una una feria en El Área Limítrofe. Esto le recordó al momento traumático de su vida en el que tuvo que conseguir trabajo en una feria, donde creó a su amigo Globito, solo para perderlo más tarde una noche triste que jamás va a olvidar.

Es por esto que ideó un plan maligno para DESTRUIR EL ÁREA LÍMITROFE, con la esperanza de que Perry el Ornitorrinco no volviera a entrometerse...

2. Objetivo

El presente trabajo práctico tiene como objetivo que el alumno:

- Diseñe y desarrolle las funcionalidades de una biblioteca con un contrato preestablecido.
- Se familiarice con y utilice correctamente los tipos de datos estructurados.

Por supuesto, se requiere que el trabajo cumpla con las buenas prácticas de programación profesadas por la cátedra.

Se considerarán críticos la modularización, reutilización y claridad del código.

3. Enunciado

Perry, el ornitorrinco agente secreto, tiene una misión crítica en la feria de la ciudad Danville. Su objetivo es desactivar bombas dispersas estratégicamente por el malvado doctor Doofenshmirtz, antes de que se arruine toda la feria y cause daños. Sin embargo, en el momento más crucial la familia Flynn decide ir a pasear por la feria, es por esto que Perry debe tener cuidado y no dejarse ver ante su familia como agente, para eso necesitará camuflarse constantemente.

Para esta primera parte del desarrollo del juego, se requerirá la **inicialización** de todos los elementos en el terreno, la **visualización** del mismo, la **implementación del movimiento del personaje principal**, Perry, y también la **habilidad** de Perry de pasar a mostrarse como agente a ornitorrinco y viceversa.

El juego consta en un terreno (la feria) donde estarán dispersas Perry, las bombas, los sombreros, las golosinas y la familia de Perry.
Al comenzar el juego, Perry tendrá 3 vidas y 100 puntos de energía.

Para inicializar, primero deberán ubicarse a Perry, luego las bombas, los sombreros, las golosinas y por último la familia de Perry en el siguiente orden: Phineas, Ferb y Candace. Ningún elemento puede inicializarse por fuera de los límites del terreno ni pisar otros elementos ya inicializados.

3.1. Herramientas

3.1.1. Sombreros

Se tendrán 3 sombreros dispersos por el terreno de forma aleatoria.

3.1.2. Golosinas

Se tendrán 5 golosinas dispersas por el terreno de forma aleatoria.

3.2. Obstáculos

3.2.1. Bombas

Se tendrán 10 bombas dispersas en el terreno de forma aleatoria. Cada bomba tendrá un timer que debe ser inicializado con un número entre 50 y 300.

3.2.2. Familia Flynn

Se tendrán a los siguiente familiares dispersos por el terreno de forma aleatoria: Phineas, Ferb y Candace (se deben inicializar en ese órden).

3.3. Terreno

El terreno será representado con una matriz de 20x20, donde estarán dispersos todos los elementos mencionados anteriormente.

3.4. Modo de juego

El personaje se podrá mover en 4 direcciones:

- **Arriba**: W
- **Abajo**: S
- **Derecha**: D
- **Izquierda**: A

Además, el personaje tendrá la habilidad de camuflarse:

- **Camuflarse**: Q

4. Especificaciones

4.1. Convenciones

- **Perry**: P.
- **Phineas**: H.
- **Ferb**: F.
- **Candace**: C.

Se deberá utilizar la siguiente convención para los obstáculos:

- **Bombas**: B.

Para las herramientas:

- **Sombreros**: S.
- **Golosinas**: G.

4.2. Funciones y procedimientos

Para poder cumplir con lo pedido, se pedirá implementar las siguientes funciones y procedimientos.

```
#ifndef __FERIA_H__
#define __FERIA_H__

#include <stdbool.h>

#define MAX_BOMBAS 20
#define MAX_HERRAMIENTAS 100
#define MAX_FAMILIARES 10

typedef struct coordenada {
    int fil;
    int col;
} coordenada_t;

typedef struct personaje {
    int vida;
    int energia;
    bool camuflado;
```

```c
    coordenada_t posicion;
} personaje_t;

typedef struct bomba {
    coordenada_t posicion;
    int timer;
    bool desactivada;
} bomba_t;

typedef struct herramienta {
    coordenada_t posicion;
    char tipo;
} herramienta_t;

typedef struct familiar {
    coordenada_t posicion;
    char inicial_nombre;
} familiar_t;

typedef struct juego {
    personaje_t perry;
    bomba_t bombas[MAX_BOMBAS];
    int tope_bombas;
    herramienta_t herramientas[MAX_HERRAMIENTAS];
    int tope_herramientas;
    familiar_t familiares[MAX_FAMILIARES];
    int tope_familiares;
} juego_t;

/*
 * Inicializará el juego, cargando toda la información inicial de Perry, los obstáculos, las
    herramientas y la familia Flynn.
 */
void inicializar_juego(juego_t* juego);

/*
 * Realizará la acción recibida por parámetro.
 * La acción recibida deberá ser válida.
 * Solo se podrá mover a Perry y camuflarlo.
 */
void realizar_jugada(juego_t* juego, char accion);

/*
 * Imprime el juego por pantalla
 */
void imprimir_terreno(juego_t juego);

/*
 * El juego se dará por ganado cuando estén todas las bombas desactivadas.
 * Si el personaje se queda sin vidas, el juego se dará por perdido.
 * Devuelve:
 * --> 1 si es ganado
 * --> -1 si es perdido
 * --> 0 si se sigue jugando
 */
int estado_juego(juego_t juego);

#endif /* __FERIA_H__ */
```

Observación: Queda a criterio del alumno/a hacer o no más funciones y/o procedimientos para resolver los problemas presentados. No se permite agregar funciones al .h presentado por la cátedra, como tampoco modificar las funciones dadas.

5. Resultado esperado

Este TP es la primera parte de un juego más complejo que van a terminar de implementar en la segunda parte. La idea es tener una base de la cual se podrá construir un juego más divertido. Aún así esperamos que el trabajo tenga una funcionalidad mínima obligatoria que cumpla con las especificaciones mencionadas anteriormente. Se deberá:

- Implementar todas las funciones especificadas en la biblioteca. Algunas todavía no podrán ser probadas dentro del juego (como estado_juego) pero su funcionamiento debería ser correcto de todas formas.
- Inicializar **todos** los campos del registro juego_t.

- Pedirle al usuario que ingrese una acción **válida** a realizar cada turno. Por el momento las acciones son moverse y camuflarse.
- Imprimir el terreno de forma clara con información que pueda serle útil al usuario (cuanta energía le queda, si está camuflado o no, etc.)
- Respetar las buenas prácticas de programación que profesamos en la cátedra.

6. Compilación y Entrega

El trabajo práctico debe ser realizado en un archivo llamado feria.c, lo que sería la implementación de la biblioteca feria.h. El trabajo debe ser compilado sin errores al correr el siguiente comando:

```
gcc *.c -o juego -std=c99 -Wall -Wconversion -Werror -lm
```

Aclaración: El main tiene que estar desarrollado en un archivo llamado juego.c, el cual manejará todo el flujo del programa.

Lo que nos permite ***.c** es agarrar todos los archivos que tengan la extensión .c en el directorio actual y los compile todos juntos. Esto permite que se puedan crear bibliotecas a criterio del alumno, aparte de las exigidas por la cátedra.

Por último debe ser entregado en la plataforma de corrección de trabajos prácticos **AlgoTrón** (patente pendiente), en la cual deberá tener la etiqueta **¡Exito!** significando que ha pasado las pruebas a las que la cátedra someterá al trabajo.

IMPORTANTE! Esto no implica necesariamente haber aprobado el trabajo ya que además será corregido por un colaborador que verificará que se cumplan las buenas prácticas de programación.

Para la entrega en **AlgoTrón** (patente pendiente), recuerde que deberá subir un archivo **zip** conteniendo únicamente los archivos antes mencionados, sin carpetas internas ni otros archivos. De lo contrario, la entrega no será validada por la plataforma.

7. Anexos

7.1. FAQ

En este link encontrarán el documento de FAQ del TP, donde se irán cargando dudas realizadas con sus respuestas. Todo lo que esté en ese documento es válido y oficial para la realización del TP.

7.2. Obtención de números aleatorios

Para obtener números aleatorios debe utilizarse la función **rand()**, la cual está disponible en la biblioteca stdlib.h.

Esta función devuelve números pseudo-aleatorios, esto quiere decir que, cuando uno ejecuta nuevamente el programa, los números, aunque aleatorios, son los mismos.

Para resolver este problema debe inicializarse una semilla, cuya función es determinar desde dónde empezarán a calcularse los números aleatorios.

Los números arrojados por **rand()** son enteros sin signo, generalmente queremos que estén acotados a un rango (queremos números aleatorios entre tal y tal). Para esto, podemos obtener el resto de la división de **rand()** por el valor máximo del rango que necesitamos.

Aquí dejamos un breve ejemplo de como obtener números aleatorios entre 10 y 30.

```c
#include <stdio.h>
#include <stdlib.h> // Para usar rand
#include <time.h>   // Para obtener una semilla desde el reloj

int main(){
        srand ((unsigned)time(NULL));
        int numero = rand() % 20 + 10; // la amplitud del rango es 20 y el valor mínimo es 10.
        printf("El valor aleatorio es: %i\n", numero);

        return 0;
}
```

7.3. Limpiar la pantalla durante la ejecución de un programa

Muchas veces nos gustaría que nuestro programa pueda verse siempre en la pantalla sin ver texto anterior.

Para esto, podemos utilizar la llamada al sistema **clear**, de esta manera, limpiaremos todo lo que hay en nuestra terminal hasta el momento y podremos dibujar la información actualizada.

Y se utiliza de la siguiente manera:

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        printf("Escribimos algo\n");
        printf("que debería\n");
        printf("desaparecer...\n");

        system("clear"); // Limpiamos la pantalla

        printf("Solo deberiamos ver esto...\n");
        return 0;
}
```

7.4. Distancia Manhattan

Para obtener la distancia entre 2 puntos mediante este método, se debe conocer a priori las coordenadas de dichos puntos.

Luego, la distancia entre ellos es la suma de los valores absolutos de las diferencias de las coordenadas. Se ve claramente en los siguientes ejemplos:

- La distancia entre los puntos (0,0) y (1,1) es 2 ya que: $|0 - 1| + |0 - 1| = 1 + 1 = 2$
- La distancia entre los puntos (10,5) y (2,12) es 15 ya que: $|10 - 2| + |5 - 12| = 8 + 7 = 15$
- La distancia entre los puntos (7,8) y (9,8) es 2 ya que: $|7 - 9| + |8 - 8| = 2 + 0 = 2$