Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE-0117, Programación Bajo Plataformas Abiertas

Proyecto final: Gin Rummy

Sharlin Hernández (B83821)
Jimena Sancho Céspedes (B87389)

8 de octubre del 2022

Índice

1. Introducción

El presente proyecto, se lleva a cabo para poner en práctrica todos los conocimientos obtenidos en el curso en la programación del lenguaje C, así como también hacer uso de herramientas para desarrolar el mismo. El lenguaje C se utiliza para el desarrollo de sistemas operativos, el cual permite desarrollar programas a bajo nivel por medio de sus características, especialmente la capacidad de manipular la memoria e invocar llamadas al sistema, como se podra apreciar en la estructura del trabajo realizado.

2. Diseño general

En esta parte del proyecto se encuentran los códigos realizados para la funcionalidad del juego Gin Rummy, así también con una breve explicación de lo que se hizo para tener un mejor entendimento del proceso y compilación del código. Es importante mencionar que las reglas y casos se tomaron del link del juego proporcionado por el docente.

Lo primero que vamos a explicar es el código del main.c; para esta parte del juego, en primer lugar, se seleccionan las bibliotecas a utilizar. Se inicia con stdio.h, la cual es la cabecera que contiene los macros, constantes, declaración de las funciones,entradas, salidas, estándar y demás. También se utiliza el time.h la cual define cuatro tipos de variables dos macros y varias funciones para manioular fechas y hora, por otra parte se tiene la stdbool.h esta define los tipos true y false; que al final son enteros representandos por el 1 y el 0 respectivamente o sea lenguaje binario. Finalmente se tiene la libgenericgame.h la cual llama a todas las librerías que ejecutan el proyecto.
Seguidamente se implementa el menú por medio de la función principal y se imprimen las opciones, dentro de las cuales se encuentran iniciar, las mejores 10 puntuaciones y la opción salir. Se ha tomado las liberrias de manera indpendiente para con ello hacer más sencillo su uso, es decir no estar llamando las funciones cada vez que se necesite.

```c
#include <stdio.h>
#include <time.h>
#include <stdbool.h>
#include "libgenericgame.h"

int main() {
    srand(time(0));
    int option_mainmenu = 0;

    do{
        printf("1- Iniciar Juego Nuevo\n"
               "2- Ver las 10 mejores puntuaciones\n"
               "3- Salir \n"
               "Inserte Opcion: ");
        scanf("%d", &option_mainmenu);
        if(inRange(1, 2, option_mainmenu)){
            if(option_mainmenu == 1){
                vec_t deck = init_deck(4, 13),
                      stack_deck = init_deck(0, 0);

                shuffle_deck(&deck);
                struct player players[2];

                for (int i = 0; i < 2; ++i)
                    players[i] = init_player();
                bool cp = rand() % 2;

                players[0].name = malloc(100 * sizeof (char));
                players[1].name = malloc(100 * sizeof (char));

                printf("Jugador1, inserte su nombre: "); scanf("%s",
                   players[0].name);
                printf("Jugador2, inserte su nombre: "); scanf("%s",
                   players[1].name);
                //Repartir Cartas
                for (int i = 0; i < 2; ++i)
                    for (int card = 0; card < 10; ++card)
                        pushback_deck(&players[i].hand, popback_deck(&
                            deck));

                int particular_option_user = 0;

                struct pcard top_stackd = popback_deck(&deck);

                pushback_deck(&stack_deck, top_stackd);
                bool SOMEONEWONTWIN = true;
                do{
                    bool current_player =  change_player(&cp);
                    card_fusion_game(players, deck, stack_deck,
                        top_stackd, current_player);
                    printf("\nTe toca (%s) Que deseas hacer?", players[
                        current_player].name);
```

```
printf("\n1-Robar del Mazo principal\n"
       "2-Robar del Mazo de descarte\n"
       "4-Anunciar Knock\n"
       "Ingrese # de opcion: ");
scanf("%d", &particular_option_user);
if(inRange(1, 4, particular_option_user)){
    switch (particular_option_user) {
        case 1:{
            struct pcard temp = popback_deck(&deck);
            pushback_deck(&players[current_player].
                hand, temp);
            cstatus_game_terminal(players, deck,
                stack_deck, top_stackd);
            card_to_discard_game(&players[
                current_player], deck, &stack_deck, &
                top_stackd);
            card_fusion_game(players, deck,
                stack_deck, top_stackd, current_player
                );
            break;
        }
        case 2:{
            if (stack_deck.size != 0)
                pushback_deck(&players[current_player
                    ].hand, top_stackd);
            top_stackd = getpcard_deck(stack_deck,
                stack_deck.size - 2);
            popback_deck(&stack_deck);
            cstatus_game_terminal(players, deck,
                stack_deck, top_stackd);
            card_to_discard_game(&players[
                current_player], deck, &stack_deck, &
                top_stackd);
            card_fusion_game(players, deck,
                stack_deck, top_stackd, current_player
                );
            break;
        }
        case 4:{
            cstatus_game_terminal(players, deck,
                stack_deck, top_stackd);
            int status = knock_alert_game(players,
                deck, stack_deck, top_stackd, &
                current_player, &SOMEONEWONTWIN);
            if (status != -9999){
                printf("Gana la Ronda %s", players[
                    current_player].name);
                saved_score_game("scoresaved.txt",
                    players[current_player].name,
                    status);
            }
            break;
```

```
                        }
                        default: break;
                    }
                }
                int scoredverified_gin = sum_current_deck(players[
                    current_player].hand);
                if( scoredverified_gin == 0){
                    printf("Gana la Ronda %s", players[current_player
                        ].name);
                    saved_score_game("scoresaved.txt", players[
                        current_player].name, sum_current_deck(players
                        [!current_player].hand) + 150);
                }
            } while (SOMEONEWONTWIN);
        }
        else show_best_score_game("scoresaved.txt");
    }
}while(option_mainmenu != 3);
return 0;
}
```

A continuación se muestra como funciona el juego con lo antes mencionado.

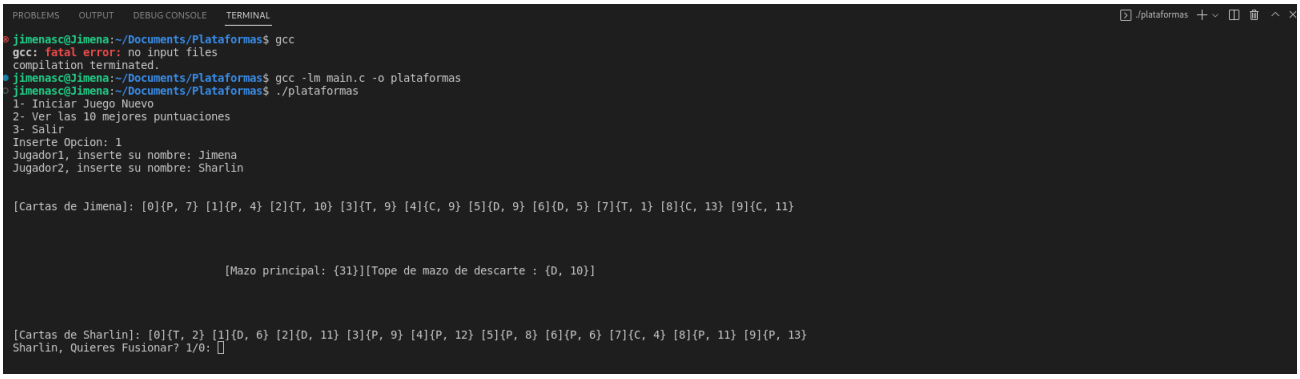

Figura 1: Muestra del código a la hora de correrlo en pantalla.

Como se observa en la figura 1, a la hora de correr el código en la terminal se muestra que se piden dos nombres de usuario los cuales van a ser los jugadores. Luego, se va a desplegar una pantalla con las cartas de cada jugador, en medio el total que hay para la partida y finalmente el tope de mazo de descarte.

2.1. Archivo de vectores (vec.h)

el archivo de vectores se creo con la finalidad de que se encargue de las operaciones que contienen los valores del juego al ser repartidas las cartas, saber cuantas cartas van quedando en el mazo conforme cada jugador "*come*".

```c
#ifndef VEC_H

#define VEC_H

#define VEC_SUCCESS 0
#define VEC_INDEX_OUT_OF_BOUNDS 1
#define VEC_COULD_NOT_ALLOCATE_MEMORY 2
#define VEC_NOT_FOUND 3
#define VEC_ALREADY_INITIALIZED 4
#define VEC_ALREADY_DESTROYED 5
#define VEC_NULL_BUFFER 6

#include <stdint.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    uint32_t allocated_slots;
    uint32_t size;
    size_t element_size;
    void * array;
} vec_t;

typedef int (*cmpfn)(const void*,const void*);

/**
 * given a pointer to a vec_t struct, and the size of
 * the elements that will be stored in the vector,
 * initializes the vector with some memory. vector must
 * have been zeroed out first, or may trigger already initialized error
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_COULD_NOT_ALLOCATE_MEMORY
 *  VEC_ALREADY_INITIALIZED
 *
 */
int init (vec_t * vector, size_t element_size);

/**
 * appends the item pointed to by the element_ptr
 * to the end of the array. grows if needed
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_COULD_NOT_ALLOCATE_MEMORY
 */
int append(vec_t * vector, void * element_ptr);

/**
 * inserts the item pointed to by the element_ptr into
 * the given index, shifting everything else over to the left.
```

```
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_COULD_NOT_ALLOCATE_MEMORY
 *  VEC_INDEX_OUT_OF_BOUNDS
 */
int insert(vec_t * vector, void * element_ptr, int idx);

/**
 * overwrites the item at the given index with the contents of the
 * element_ptr buffer. Be sure to not leak memory when using this!
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_INDEX_OUT_OF_BOUNDS
 */
int replace(vec_t * vector, void * element_ptr, int idx);

/**
 * returns the current length of the vector
 */
__attribute__((unused)) int veclen(vec_t * vector);

/**
 * removes the given item that is equivalent to the
 * element pointed to by element_ptr and shifts everything
 * else over to the left. Shrinks the array
 * if less than 1/4 of the allocated space is in use.
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_COULD_NOT_ALLOCATE_MEMORY
 *  VEC_NULL_BUFFER
 *  VEC_NOT_FOUND
 */
int remove_element(vec_t * vector, void * element_ptr);

/**
 * removes the given item that is at the given index
 * and shifts everything else over to the left. Shrinks the array
 * if less than 1/4 of the allocated space is in use.
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_COULD_NOT_ALLOCATE_MEMORY
 *  VEC_INDEX_OUT_OF_BOUNDS
 */
int remove_index(vec_t * vector, int idx);

/**
 * gets the item at the given index and copies it into
 * the memory pointed to by element_buffer
```

```
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_INDEX_OUT_OF_BOUNDS
 *  VEC_NULL_BUFFER
 */
int get(vec_t * vector, int idx, void * element_buffer);

/**
 * frees all memory given to this vector
 *
 * possible return values:
 *  VEC_SUCCESS
 *  VEC_ALREADY_DESTROYED
 */
int destroy(vec_t * vector);

/**
 * sorts the array in place. Always succeeds.
 *
 * possible return values:
 *  VEC_SUCCESS
 */
int sort(vec_t * vector,cmpfn cmp);
#endif
```

2.2. Archivo de cartas (pcard)

En esta sección se tiene el tipo y número de carta para los cuales se ha utilizado, rey (K), reina (Q), jota (J), Diamante(D) y los números del as al diez.

```
#ifndef SHARLIN_PCARDLIB_H
#define SHARLIN_PCARDLIB_H
#include <assert.h>
#include "utilsfun.h"
struct pcard{
    char type;
    int number;
};
struct pcard _pcard(char _type, int _number);
#endif //SHARLIN_PCARDLIB_H
int pcard_comparator(void *pc1, void *pc2);
```

2.3. Archivo de jugador (player.h)

Va toda la estructura relacionada a la información del los jugadores, como lo son el nombre, la mano y la mesa de juego.

Para estos archivos se ha utilizado el struct para definir su tiempo y los char e int para el resto de la informacón de acuerdo a su conenido.

```
#ifndef SHARLIN_PLAYERLIB_H
#define SHARLIN_PLAYERLIB_H
#include "vec.h"
#include "pcardlib.h"
struct player{
    char *name;
    vec_t hand;
    vec_t *main_table;
    int cant_fusions;
};

bool change_player(bool * current_player);
struct player init_player();
#endif //SHARLIN_PLAYERLIB_H
```

2.4. Archivo de calificaciones (utilsfun.h)

En esta sección se trabajo con datos de tipo boolean, debido a que intermante es la forma en la que se van a desarrollar las calificaciones, esta estructura se conforma en su mayoria de la función void con punteros, que iran almacenado y arrojando las calificaciones. Además se ha utilizado un archivo .text para colocar los nombres de los jugadores con sus puntajes, es decir como un tipo de base de datos de los mismos.

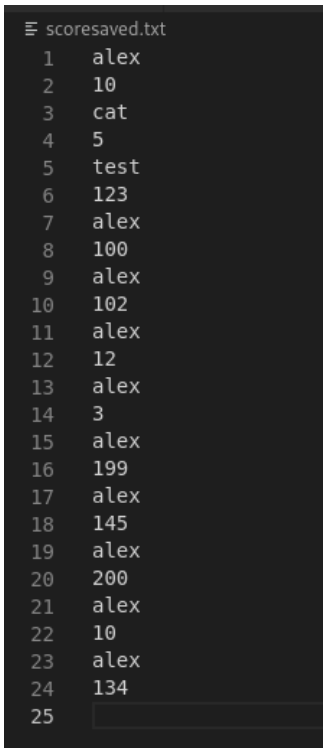

Figura 2: estructura de calificaciones.

3. Principales retos

Nuestros principales retos como equipo fueron que al no tener un buen entendimiento de la funcionalidad del Gin Rummy, ya que antes jamás lo habíamos jugado primero tuvimos que investigar sus reglas, videos tutoriales en youtube. Todo esto para comprender a mayor profundidad y poder programar el juego. El segundo reto fue que al no tener un buen manejo de ciertos temas de c, se tuvo que volver a ver los videos del profesor en mediación virtual y algunos tutoriales en youtube ya que, al estar indagando tanto encontramos más estructurales y funciones que no conocíamos. Otro reto que tuvimos como grupo fue que las máquinas virtuales no nos querían funcionar de la manera correcta por lo que, al no tener una buena plataforma en donde se pudiera trabajar se buscaron alterativas como: instalar el visual studio code desde nuestro sistema operativo general (Windows) y también instalar git de la misma manera ya que, cuando se trataba de subir los archivos desde la terminal de Linux salía un error de contraseña del usuario root que no se pudo resolver. Además, el factor tiempo jugó en contra nuestra debido a que, al no conocer tanto de C y sumando el final de semestre se nos complicó encontrar tiempo para reunirnos e ir avanzando en el trabajo. Finalmente, al ser solo dos personas como integrantes del grupo hizo que el proyecto se sintiera más pesado porque si fueramos al menos tres el trabajo hubiera sido más óptimo y no se habría perdido tanto tiempo intentando de entender como funciona el Gin Rummy.

4. Conclusiones

Se concluye que por medio del presente trabajo se puede poner en práctica cada uno de los temas vistos en clase, de manera tal que por su estructura es necesario usar la mayoria de conetnidos, así mismo se puede evidenciar la importancia y la utilidad de las bibliotecas para hacer posible el funcinamiento del proyecto.

Por otra parte, con el proyecto se pudo poner en práctica posibles retos a los cuales un profesional en la ingenería se debe enfrentar en el area de la programación, tomando en consideracíon no solo la creación de un código que funcione según los requerimientos de un cliente sino también brindar documentación teoríca de como funciona lo que se hizo.