SEMESTER PROJECT

NAME: TRAȘCĂ ANDREI-SABIN

GROUP: 422G

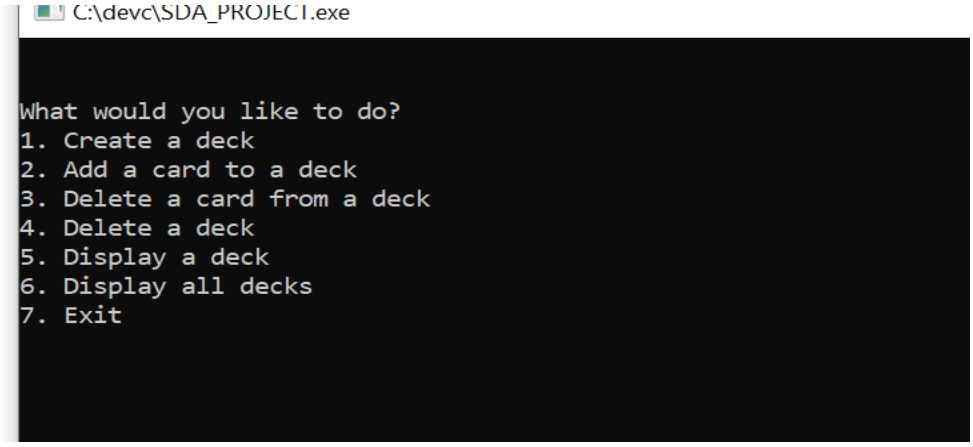
1.Description of the program in natural language

The program simulates a deck of playing cards. Using a user interface, it allows us to create a new deck of cards, add cards (having values from 1 to 14 and the following symbols: clubs, hearts, spades, diamonds). We can also delete a specified deck or a specified card from a specified deck. In the end, we can display a specified deck or all the decks that have been created.

In order to achieve the problem demands, I have used doubled linked lists , 2 structures(one for the card attributes and one for the deck attributes) and multiple functions.

2.Instances of running the program(screenshots)

The first requirement of the program is to introduce from the keyboard the user input.



Let's say we chose the "1" option. It will take us to a menu where we must specify what number is the deck we are creating. Then we will be asked to introduce the number of cards we wish to add to the deck and also their values (1-14) and symbols (clubs, hearts, diamonds, spades).Once created the program will display "The deck has been created".

Then, let's say we want to add a card to the current deck. We will select option "2" and then we will be asked to type the number of the deck we wish to add the card to and also the value and symbol of the card. When the process is done we will receive a message "Card added!".

```
2
Enter deck number: 1
Enter card value and symbol (e.g. 7 clubs): 12 spades

CARD ADDED!
```

Now, let's add another deck using option "1".

```
1
Enter deck number: 2

Enter number of cards to add to deck: 5
Enter card value and symbol (e.g. 7 clubs): 1 diamonds

CARD ADDED!
Enter card value and symbol (e.g. 7 clubs): 1 clubs

CARD ADDED!
Enter card value and symbol (e.g. 7 clubs): 3 spades

CARD ADDED!
Enter card value and symbol (e.g. 7 clubs): 14 hearts

CARD ADDED!
Enter card value and symbol (e.g. 7 clubs): 7 diamonds

CARD ADDED!

THE DECK HAS BEEN CREATED!
```

Now, let's use option "3" to delete a card from deck number 2. We will introduce from the keyboard the deck from where we delete the card, then the value and the symbol of the card we wish to delete. After the process is complete a message will show saying "Card deleted!".

```
3
Enter deck number: 2
Enter card value: 1
Enter card symbol: clubs

CARD DELETED!
```

Now, let's use option "6" to display all the decks we have created and modified.

```
6
Deck Number: 1
5 of diamonds
10 of hearts
9 of clubs
12 of spades
Deck Number: 2
1 of diamonds
3 of spades
14 of hearts
7 of diamonds
```

If we are done with our program, we can now use option "7" to clear our memory and exit the program.

```
7

THE DECK HAS BEEN DELETED!

THE DECK HAS BEEN DELETED!

--------------------------------
Process exited after 1144 seconds with return value 0
Press any key to continue . . .
```

3.The listing for entire program (with useful comments)

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

typedef struct card {
   int value;
   char symbol[10];
   struct card* next;
   struct card* prev;
} card;

typedef struct deck {
   int number;
   card* head;
   card* tail;
   struct deck* next;
   struct deck* prev;
} deck;

deck* create_deck(int number) {
            //function for creating a new deck//
            int i;
   deck* new_deck = (deck*) malloc(sizeof(deck));
   new_deck->number = number;
   new_deck->head = NULL;
   new_deck->tail = NULL;
   new_deck->next = NULL;
   new_deck->prev = NULL;
   int num_cards;
   printf("Enter number of cards to add to deck: ");
   scanf("%d", &num_cards);
   for(i = 0; i < num_cards; i++) {
      add_card(new_deck);
   }
   printf("\nTHE DECK HAS BEEN CREATED!\n");
   return new_deck;
}


void add_card(deck* d) {
            //adding cards to a existing deck//
   card* new_card = (card*) malloc(sizeof(card));
   int value;
   char symbol[10];
   printf("Enter card value and symbol (e.g. 7 clubs): ");
   scanf("%d %s", &value, symbol);
   //checking if the valuw is between 1-14//
   if(value < 1 || value > 14){
      printf("The provided card value is not a valid one\n");
      return;
   }
```

```
    //checking if the symbol is within the acceptable ones//
    if(strcmp(symbol,"diamonds") && strcmp(symbol,"clubs")&& strcmp(symbol,"hearts")&&
strcmp(symbol,"spades")){
        printf("The provided card symbol is not a valid one\n");
        return;
    }
    new_card->value = value;
    strcpy(new_card->symbol,symbol);
    new_card->next = NULL;
    new_card->prev = d->tail;
    if(d->tail) {
        d->tail->next = new_card;
    }
    d->tail = new_card;
    if(!d->head) {
        d->head = new_card;
    }
    printf("\nCARD ADDED!\n");
}


void delete_card(deck* d, card* c) {
            //function from deleting a card//
    if(c->prev) {
        c->prev->next = c->next;
    }
    if(c->next) {
        c->next->prev = c->prev;
    }
    if(c == d->head) {
        d->head = c->next;
    }
    if(c == d->tail) {
        d->tail = c->prev;
    }
    free(c);
    printf("\nCARD DELETED!\n");
}

void delete_deck(deck* d) {
            //function for deleting a deck//
    card* c = d->head;
    while(c) {
        card* next = c->next;
        free(c);
        c = next;
    }
    free(d);
    printf("\nTHE DECK HAS BEEN DELETED!\n");
}

void display_deck(deck* d) {
            //function for displaying a specified deck//
    printf("Deck Number: %d\n", d->number);
    card* c = d->head;
    while(c) {
```

```
        printf("%d of %s\n", c->value, c->symbol);
        c = c->next;
    }
}

void display_deck_cards(deck* d) {
            //function for displaying the cards within a specidied deck//
    card* c = d->head;
    while(c) {
        printf("%d of %s\n", c->value, c->symbol);
        c = c->next;
    }
}

void display_all(deck* head) {
            //fucntion for displaying all the available decks//
    deck* current = head;
    while(current) {
        printf("Deck Number: %d\n", current->number);
        display_deck_cards(current);
        current = current->next;
    }
}


int main() {
    deck* head = NULL;
    deck* tail = NULL;
    int user_choice;
    do{     printf("\n\n");
        printf("What would you like to do?\n");
        printf("1. Create a deck\n");
        printf("2. Add a card to a deck\n");
        printf("3. Delete a card from a deck\n");
        printf("4. Delete a deck\n");
        printf("5. Display a deck\n");
        printf("6. Display all decks\n");
        printf("7. Exit\n\n");
        scanf("%d", &user_choice);
        switch(user_choice) {
            case 1: {
                int deck_num;
                printf("Enter deck number: ");
                scanf("%d", &deck_num);
                printf("\n");
                deck* new_deck = create_deck(deck_num);
                if(!head) {
                    head = new_deck;
                    tail = new_deck;
                } else {
                    new_deck->prev = tail;
                    tail->next = new_deck;
                    tail = new_deck;
                }
                break;
            }
```

```
case 2: {
    int deck_num;
    printf("Enter deck number: ");
    scanf("%d", &deck_num);
    deck* current = head;
    while(current) {
                                                                    if(current->number == deck_num) {
            add_card(current);
            break;
        }
        current = current->next;
    }
    if(!current) {
        printf("Deck not found\n");
    }
    break;
}
case 3: {
    int deck_num, card_value;
    char card_symbol[10];
    printf("Enter deck number: ");
    scanf("%d", &deck_num);
    printf("Enter card value: ");
    scanf("%d", &card_value);
    printf("Enter card symbol: ");
    scanf("%s", card_symbol);
    deck* current = head;
    while(current) {
        if(current->number == deck_num) {
            card* c = current->head;
            while(c) {
                if(c->value == card_value && !strcmp(c->symbol, card_symbol)) {
                    delete_card(current, c);
                    break;
                }
                c = c->next;
            }
            if(!c) {
                printf("Card not found in deck\n");
            }
            break;
        }
        current = current->next;
    }
    if(!current) {
        printf("Deck not found\n");
    }
    break;
}
case 4: {
    int deck_num;
    printf("Enter deck number: ");
    scanf("%d", &deck_num);
    deck* current = head;
```

```
    while(current) {
        if(current->number == deck_num) {
            if(current == head) {
                head = current->next;
                if(head) {
                    head->prev = NULL;
                }
            }
            if(current == tail) {
                tail = current->prev;
                if(tail) {
                    tail->next = NULL;
                }
            }
            if(current->prev) {
                current->prev->next = current->next;
            }
            if(current->next) {
                current->next->prev = current->prev;
            }
            delete_deck(current);
            break;
        }
        current = current->next;
    }
    if(!current) {
        printf("Deck not found\n");
    }
    break;
}
case 5: {
    int deck_num;
    printf("Enter deck number: ");
    scanf("%d", &deck_num);
    deck* current = head;
    while(current) {
        if(current->number == deck_num) {
            display_deck(current);
            break;
        }
        current = current->next;
    }
    if(!current) {
        printf("Deck not found\n");
    }
    break;
}
case 6: {
    display_all(head);
    break;
}
case 7: {
    break;
}
default: {
                                    printf("Invalid option\n");
```

```
            break;
          }
        }
    } while(user_choice != 7);
    deck* current = head;
    while(current) {
        delete_deck(current);
        current = current->next;
    }
    return 0;
}
```

4.Workflow for every function

deck create_deck(int number)*

- It creates a new deck structure and allocates memory for it using malloc().
- The new deck's number, head, tail, next, and prev pointers are set to the passed in number, NULL, NULL, NULL, and NULL, respectively.
- The user is prompted to enter the number of cards to add to the deck.
- The function then uses a for loop to call the "add_card" function as many times as the number of cards entered by the user.
- Each time the "add_card" function is called, it adds a new card to the deck.
- After all cards have been added, a message is printed to indicate that the deck has been created.
- The function then returns a pointer to the new deck

void add_card(deck d)*

- The function creates a new card structure and allocates memory for it using malloc().
- The user is prompted to enter the value and symbol for the card.

- The entered value is checked to make sure it is between 1 and 14. If it is not, an error message is printed and the function returns.
- The entered symbol is checked to make sure it matches one of the acceptable symbols (diamonds, clubs, hearts, or spades). If it does not, an error message is printed and the function returns.
- The new card's value and symbol are set to the entered values.
- The new card's next and previous pointers are set to NULL and the current tail of the deck, respectively.
- If the deck has a tail, the tail's next pointer is set to the new card.
- The new card is set as the new tail of the deck.
- If the deck does not have a head, the new card is set as the head of the deck.
- A message is printed to indicate that the card has been added.

void delete_card(deck d, card* c)*

- The function checks if the previous card in the deck is not null, if true, it sets the next pointer of the previous card to the next pointer of the current card.
- The function checks if the next card in the deck is not null, if true, it sets the previous pointer of the next card to the previous pointer of the current card.
- The function checks if the current card is the head of the deck, if true, it sets the head of the deck to the next card.
- The function checks if the current card is the tail of the deck, if true, it sets the tail of the deck to the previous card.
- The function frees the memory allocated to the current card.
- A message is printed to indicate that the card has been deleted.

void delete_deck(deck d)*

- The function sets a pointer "c" to the head of the deck.
- The function enters a while loop that continues as long as "c" is not null.
- Within the while loop, the function sets a pointer "next" to the next card in the deck.
- The function frees the memory allocated to the current card.
- The function sets "c" to "next" to move to the next card in the deck.
- Once the loop has completed, the function frees the memory allocated to the deck structure.
- A message is printed to indicate that the deck has been deleted.

void display_deck(deck d)*

- The function prints the number of the deck using the "number" field of the deck structure.
- The function sets a pointer "c" to the head of the deck.
- The function enters a while loop that continues as long as "c" is not null.
- Within the while loop, the function prints the value and symbol of the current card using the "value" and "symbol" fields of the card structure.
- The function sets "c" to "c->next" to move to the next card in the deck.
- Once the loop has completed, the function exits.

void display_all(deck head)*

- The function sets a pointer "current" to the head of the list of decks.

- The function enters a while loop that continues as long as "current" is not null.
- Within the while loop, the function prints the number of the current deck using the "number" field of the deck structure.
- The function calls the "display_deck_cards" function, passing in the current deck, which will display the cards in the current deck.
- The function sets "current" to "current->next" to move to the next deck in the list.
- Once the loop has completed, the function exits.