War_game.cpp

User manual

Program: main.cpp

- This programs replicates the card game called war.
- Computer acts as both, the player 1 and also as the player 2.
- Once the game starts, the computer plays the game and prints out the cards that both of the user played for each particular round.
- At the end, winner player is printed to the console.

System Manual

Program: test_socialite.cpp

Main function:
int main()

- This programs replicates the card game called war.
- Computer acts as the player 1 and also as the player 2.
- This function first calls the cre_deck() method of the class deck which creates a deck of card for the game.
- The suffle_deck() method of class deck is called that suffles the card deck.
- Then div_deck() method of class deck is called that divides the deck of 52 cards into among two players (both of which is computer in this case).
- Once each player gets their deck of card, game_start() function is contains code for the game play.
- After the game is over, the winner() function is called to check which player won the game.

int game_start()

- This function is responsible for the entire game play.
- It keeps drawing a single card from each player's deck and also decides which player won each round.
- In case if anyone or both players runs out of the card, this function returns 0, indicating that the game is over.
- First a card is drawn from both the player's deck.
- Card drawn from player 1's deck is stored in temp_1 object of card class.
- And card drawn from player 2's deck is stored in temp_2 object of card class.
- If player 1's card is of greater rank than player 2's card, both the card is added to player 1's deck.
- Else if player 2's card is of greater rank than player 1's card, both the card is added to player 2's deck.
- In case if both the cards are of same rank, then there is a "war".
- In this case, if both of the players has at least 2 cards, then two cards are drawn from deck of each player.
- The 2 cards drawn from player 1's deck is stored in object temp_1_a of deck data type.
- The 2 cards drawn from player 2's deck is stored in object temp_2_a of deck data type.
- Cards in both of these decks are also added to a temporary deck object called "total".
- Then check_war function is called to decide the winner in case of war. Both the deck (temp_1_a and temp_2_a) is passed to this function as arguments.
- check_war() function returns 1 if the winner is player 1, 2 if the winner is player 2 and 0 if any one of the player runs out of card during war.
- If the value returned by check_war() function is 1, the cards stored in total is added to player 1's deck. Else if the value returned by check_war() function is 2, the cards stored in total is added to

player 2's deck. The object "total" is cleared in both of these cases. And if the value returned from check_war() function is 0, the cards is not added to anyone's deck and the function is terminated.

int check_war()

- This function checks which player wins the round during a war.
- It takes two deck of card as its parameters. One deck contains the 2 cards drawn from player 1's deck during a war. And the other deck contains the other 2 cards drawn from the deck of player 2 during the war.
- The top card from the deck of player 1 that was passed as an argument to this function during the war is then stored in the temp object of card class named "a".
- The top card from the deck of player 2 that was passed as an argument to this function during the war is then stored in the temp object of card class named "b".
- a is then compared with b
- If card a has higher rank than card b, 1 is returned by this function indicating that player 1 won.
- If card b has higher rank than card a, 2 is returned by this function indicating that player 2 won.
- In case if the cards are equal, then the function checks weather both the players has at least 2 cards remaining in their deck or not, if no then 0 is returned indicating that the game is over and either one or both of the player does not have enough card required for the war.
- In case if both the player has at least 2 cards, then the war is carried out again.
- 2 cards are drawn from the deck of each player.
- Cards drawn from player 1's deck is stored in temp_1_a object of deck class and cards drawn from player 2's deck is stored in temp_2_a object of deck class.
- Both of these deck (temp_1_a and temp_2_a are added to the object "total".
- The value returned by calling this function again is then returned. (Recursive call)
- This time temp_1_a and temp_2_a is passed as argument to this function itself.

int winner()

- This function decides the winner of the game.
- If both of the players has 0 or 1 cards remaining in their deck, then it's a tie. In this case "TIE!" is printed to the console and then 0 is returned to terminate this function.
- Else if player 1 has 0 card and player 2 has more than 0 card or if player 1 has 1 card and player 2 has more than 1 card, then player 2 is the winner. In this case "Player 2 Wins!" is printed to the console and 0 is returned to terminate the function.
- Else if player 2 has 0 card and player 1 has more than 0 card or if player 2 has 1 card and player 1 has more than 1 card, then player 1 is the winner. In this case "Player 1 Wins!" is printed to the console and 0 is returned to terminate the function.

Deck header file

deck.h

- This header file first checks if _DECK_H_ has been defined or not. If it is not defined, then the header file _ DECK_H_ is defined.
- This file then contains the deck class definition.
- The class declared in this header file contains only the prototype of the methods and the attributes required for this class.
- All the methods are defined to be public and the variables/ attributes are defined to be private.

Accessibility	Type of method	Prototype	Use
Public	Constructor	`deck()`	
	Inspector	`card deal_single()`	Returns a single card from the deck
		`deck deal_multi(int)`	Returns more than one card (a deck) from a deck
		`card get_card(int i) const`	Returns the card stored in i'th position of the all_card vecctor
		`int num_cards() const`	Returns number of cards in a deck
	Mutators	`void cre_deck()`	To create a deck of 52 card
		`void div_deck(deck & pla_one, deck & pla_two)`	Divides a deck of 52 card into two decks of equal cards
		`void suffle_deck()`	Shuffles the deck of card
		`void add_new(card)`	Adds a card to the deck
		`void clear()`	Deletes all the cards in the deck.
Private	Attributes	`vector<card> all_card`	To store cards in the deck
Other Functions		`deck operator+(const deck & one, const deck & two)`	To overload "+" operator for deck objects

Deck class implementation

deck.cpp

- This header file has body of all the methods included in deck.h.
- The constructor calls clear method function of this class every time a new object is created.
- The cre_deck method is used to create a deck of 52 cards. It uses the setshape method of class card to set the shape of the card and the setnumber method of class card to set the rank of the class. Then it adds this card to the all_card vector that contains all the cards in the deck.
- The div_deck method is used to divide the deck of card among two players. It takes two deck objects as it's parameter. The first deck object is used to store deck of 32 cards for player 1 and the second parameter is used to store deck of 32 cards for player 2. The first half of the deck is assigned to player 1 and the second half of the deck is assigned to player 2.
- The suffle_deck() method is used to shuffle the deck of card. It shuffles the deck of card based on Fisher-Yates Shuffle.
- The deal_single() method is used to return a single card from the top of the deck of the card. It also deletes the card from the deck after it returns the card.
- The deal_multi() method is used to return a deck of card. It takes an integer as its parameter. This integer represent the number of cards that the deck should contain. It also deletes the cards from the deck that are to be returned.
- The num_cards() method returns the number of cards in the deck.
- The add_new() method is used to add a card to the bottom of the deck. It takes a card object as its's parameter. The card is the card that needs to be added to the bottom of the deck.
- The get_card() method returns the card stored in i'th location of the deck starting from the bottom of the deck.
- The clear() method deletes all the cards stored in the deck of the card.
- The operator+ is used to overload the "+" operator for the objects of deck class. This function first add all the cards in first object to a temporary deck object named temp. Then it adds all the cards in second deck object to the temp card and then returns the temp variable.

Class header file

card.h

- This header file contains definition of card class
- This header file first checks if _CARD_H_ has been defined or not. If it is not defined, then the header file _ CARD_H_ is defined.
- This file then contains the deck class definition.
- The class declared in this header file contains only the prototype of the methods and the attributes required for this class.
- All the methods are defined to be public and the variables/ attributes are defined to be private.

Accessibility	Type of method	Prototype	Use
Public	Constructor	`card()`	
	Inspector	`Suit getshape()`	Returns suit of the card
		`Rank getnumber()`	Returns rank of the card
	Mutators	`void setshape(Suit)`	Set suit of the card
		`void setnumber(Rank)`	Set rank of the card
	Facilitators	`string convert_shape(Suit)`	Convert suit to it's equivalent alphabet
		`string convert_number(Rank)`	Convert rank to it's equivalent alphabet/ number
Private	Attributes	`Suit shape`	Stores suit of the card
		`Rank number`	Stores Rank of the card
	Operator Overload	`ostream & operator<<(ostream & out, card & ca)`	To overload "<<" operator for card objects
		`bool operator==(card one, card two)`	To compare weather two cards are equal or not
		`bool operator<(card one, card two)`	To check if first card is greater than second card
		`bool operator>(card one, card two)`	To check if first card is smaller than second card

Card class implementation

card.cpp

- This header file has body of all the methods included in card.h.
- The constructor sets shape and number to HEARTS and ACE respectively.
- The getshape() method and getnumber() method returns shape and number of the card respectively.
- The setshape() mthod and setnumber() method sets the shape and number of the card respectively.
- The convert_shape method Convert suit to it's equivalent alphabet. It takes the suit of a card as it's parameter. And returns "C" if the shape of the card is CLUBS, "S" if the shape of the card is SPADES, "D" if the shape of the card is DIAMONDS, "H" if the shape of the card is HEARTS.
- The convert_number() method converts rank of a card to it's equivalent alphabet/number. It takes rank of the card as it's parameter. And returns the alphabet/number corresponding to that rank.
- The cre_deck method is used to create a deck of 52 cards. It uses the setshape method of class card to set the shape of the card and the setnumber method of class card to set the rank of the class. Then it adds this card to the all_card vector that contains all the cards in the deck.
- The div_deck method is used to divide the deck of card among two players. It takes two deck objects as it's parameter. The first deck object is used to store deck of 32 cards for player 1 and the second parameter is used to store deck of 32 cards for player 2. The first half of the deck is assigned to player 1 and the second half of the deck is assigned to player 2.
- The suffle_deck() method is used to shuffle the deck of card. It shuffles the deck of card based on Fisher-Yates Shuffle.
- The deal_single() method is used to return a single card from the top of the deck of the card. It also deletes the card from the deck after it returns the card.
- The deal_multi() method is used to return a deck of card. It takes an integer as its parameter. This integer represent the number of cards that the deck should contain. It also deletes the cards from the deck that are to be returned.
- The num_cards() method returns the number of cards in the deck.
- The add_new() method is used to add a card to the bottom of the deck. It takes a card object as its's parameter. The card is the card that needs to be added to the bottom of the deck.
- The get_card() method returns the card stored in i'th location of the deck starting from the bottom of the deck.
- The clear() method deletes all the cards stored in the deck of the card.
- The operator+ is used to overload the "+" operator for the objects of deck class. This function first add all the cards in first object to a temporary deck object named temp. Then it adds all the cards in second deck object to the temp card and then returns the temp variable.