functions () test cases

#1 int generatePlaceTable (int card, int adder)
Function description: returns ***place*** -- a 3-digit number;
the hundreds place being the table position;
the tens and ones are the card number

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	12th card, 1st cell TableCard = generatePlaceTable (12, 1);	card = 12 adder = 1 TableCard = ?	card = 12 adder = 1 TableCard = 112	card = 12 adder = 1 TableCard = 112	**P**
2	34th card, 2nd cell TableCard = generatePlaceTable (34, 2);	card = 34 adder = 2 TableCard = ?	card = 34 adder = 2 TableCard = 234	card = 34 adder = 2 TableCard = 234	**P**
3	23rd card, 5th cell TableCard = generatePlaceTable (23, 5);	card = 23 adder = 5 TableCard = ?	card = 23 adder = 5 TableCard = 523	card = 23 adder = 5 TableCard = 523	**P**
4	0, 9th cell TableCard = generatePlaceTable (0, 9);	card = 0 adder = 9 TableCard = ?	card = 0 adder = 9 TableCard = 900	card = 0 adder = 9 TableCard = 900	**P**

#2 int extractPlaceTable (int placetable)
Function description: returns ***extract*** -- the hundreds digit which defines the place in the table

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	1st cell Place = extractPlaceTable (112);	placetable = 112 Place = ?	placetable = 112 Place = 1	placetable = 112 Place = 1	**P**
2	2nd cell Place = extractPlaceTable (234);	placetable = 234 Place = ?	placetable = 234 Place = 2	placetable = 234 Place = 2	**P**
3	5th cell Place = extractPlaceTable (523);	placetable = 523 Place = ?	placetable = 523 Place = 5	placetable = 523 Place = 5	**P**
4	9th cell Place = extractPlaceTable (951);	placetable = 951 Place = ?	placetable = 951 Place = 9	placetable = 951 Place = 9	**P**

#3 int extractCardNumber (int placetable)
Function description: returns ***extract*** -- the 2 rightmost digits (ones and tens) which defines the card number

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	12th card Card = extractCardNumber (112);	placetable = 112 Card = ?	placetable = 112 Card = 12	placetable = 112 Card = 12	**P**
2	34th card Card = extractCardNumber (234);	placetable = 234 Card = ?	placetable = 234 Card = 34	placetable = 234 Card = 34	**P**
3	23rd card Card = extractCardNumber (523);	placetable = 523 Card = ?	placetable = 523 Card = 23	placetable = 523 Card = 23	**P**
4	0 Card = extractCardNumber (900);	placetable = 900 Card = ?	placetable = 900 Card = 0	placetable = 900 Card = 0	**P**

#4 int isCardJack (int card)
Function description: returns ***res*** -- 1 if Jack, returns 0 if not

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	11th card Jack = isCardJack (11);	card = 11 Jack = 0	card = 11 Jack = 1	card = 11 Jack = 1	**P**
2	50th card Jack = isCardJack (50);	card = 50 Jack = 0	card = 50 Jack = 1	card = 50 Jack = 1	**P**
3	42nd card Jack = isCardJack (42);	card = 42 Jack = 0	card = 42 Jack = 0	card = 42 Jack = 0	**P**
4	0 Jack = isCardJack (0);	card = 0 Jack = 0	card = 0 Jack = 0	card = 0 Jack = 0	**P**

#5 int isCardQueen (int card)
Function description: returns ***res*** -- 1 if Queen, returns 0 if not

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	12th card Queen = isCardQueen (12);	card = 12 Queen = 0	card = 12 Queen = 1	card = 12 Queen = 1	**P**

2	38th card Queen = isCardQueen (38);	card = 38 Queen = 0	card = 38 Queen = 1	card = 38 Queen = 1	**P**
3	30th card Queen = isCardQueen (30);	card = 30 Queen = 0	card = 30 Queen = 0	card = 30 Queen = 0	**P**
4	0 Queen = isCardQueen (0);	card = 0 Queen = 0	card = 0 Queen = 0	card = 0 Queen = 0	**P**

#6 int isCardKing (int card)
Function description: returns ***res*** -- 1 if King, returns 0 if not

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	13th card King = isCardKing (13);	card = 13 King = 0	card = 13 King = 1	card = 12 King = 1	**P**
2	26th card King = isCardKing (26);	card = 26 King = 0	card = 26 King = 1	card = 38 King = 1	**P**
3	32nd card King = isCardKing (32);	card = 32 King = 0	card = 32 King = 0	card = 30 King = 0	**P**
4	0 King = isCardKing (0);	card = 0 King = 0	card = 0 King = 0	card = 0 King = 0	**P**

#7 void identifyCardSuit (int card, char* code)
Function description: stores the card suit code in ****code***
*code is set to null by default

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	card is < 13 identifyCardSuit (10, &CardSuit);	card = 10 *code = &CardSuit = '\0'	card = 10 *code = &CardSuit = 'D'	card = 10 *code = &CardSuit = 'D'	**P**
2	card is = 13 identifyCardSuit (13, &CardSuit);	card = 13 *code = &CardSuit = '\0'	card = 13 *code = &CardSuit = 'D'	card = 13 *code = &CardSuit = 'D'	**P**
3	card is < 26 identifyCardSuit (15, &CardSuit);	card = 15 *code = &CardSuit = '\0'	card = 15 *code = &CardSuit = 'C'	card = 15 *code = &CardSuit = 'C'	**P**
4	card is = 26 identifyCardSuit (26, &CardSuit);	card = 26 *code = &CardSuit = '\0'	card = 26 *code = &CardSuit = 'C'	card = 26 *code = &CardSuit = 'C'	**P**
5	card is < 39 identifyCardSuit (33, &CardSuit);	card = 33 *code = &CardSuit = '\0'	card = 33 *code = &CardSuit = '\H'	card = 33 *code = &CardSuit = '\H'	**P**

6	card is = 39 identifyCardSuit (39, &CardSuit);	card = 39 *code = &CardSuit = '\0'	card = 39 *code = &CardSuit = '\H'	card = 39 *code = &CardSuit = '\H'	**P**
7	card is < 52 identifyCardSuit (49, &CardSuit);	card = 49 *code = &CardSuit = '\0'	card = 49 *code = &CardSuit = '\S'	card = 49 *code = &CardSuit = '\S'	**P**
8	card is = 52 identifyCardSuit (52, &CardSuit);	card = 52 *code = &CardSuit = '\0'	card = 52 *code = &CardSuit = '\S'	card = 52 *code = &CardSuit = '\S'	**P**
9	card is = 0 identifyCardSuit (0, &CardSuit);	card = 0 *code = &CardSuit = '\0'	card = 0 *code = &CardSuit = ' ' *(space)*	card = 0 *code = &CardSuit = ' ' *(space)*	**P**

#8 void identifyCardRank (int card, char* cardrank1, char* cardrank2)
Function description: stores the char code for the first character in ****cardrank1*** and
stores the char code for the second character in ****cardrank2***

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	(card - 1) % 13 == 0 identifyCardRank (14, &CardCode1, &CardCode2);	card = 14 *cardrank1 = &CardCode1 = ? *cardrank2 = &CardCode2 = ?	card = 14 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'A'	card = 14 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'A'	**P**
2	card % 13 > 1 && card % 13 < 10 identifyCardRank (45, &CardCode1, &CardCode2);	card = 45 *cardrank1 = &CardCode1 = ? *cardrank2 = &CardCode2 = ?	card = 45 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = '6'	card = 45 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = '6'	**P**
3	card % 13 == 0 identifyCardRank (39, &CardCode1, &CardCode2);	card = 39 *cardrank1 = &CardCode1 = ? *cardrank2 = &CardCode2 = ?	card = 39 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'K'	card = 39 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'K'	**P**
4	(card + 1) % 13 == 0 identifyCardRank (12, &CardCode1, &CardCode2);	card = 12 *cardrank1 = &CardCode1 = ? *cardrank2 = &CardCode2 = ?	card = 12 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'Q'	card = 12 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'Q'	**P**
5	10 *//else* identifyCardRank (10, &CardCode1, &CardCode2);	card = 10 *cardrank1 = &CardCode1 = ? *cardrank2 = &CardCode2 = ?	card = 10 *cardrank1 = &CardCode1 = '1' *cardrank2 = &CardCode2 = '0'	card = 10 *cardrank1 = &CardCode1 = '1' *cardrank2 = &CardCode2 = '0'	**P**
6	(card + 2) % 13 == 0 identifyCardRank (50, &CardCode1, &CardCode2);	card = 50 *cardrank1 = &CardCode1 = ? *cardrank2 = &CardCode2 = ?	card = 50 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'J'	card = 50 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = 'J'	**P**

7	0 identifyCardRank (0, &CardCode1, &CardCode2);	card = 0 *cardrank1 = &CardCode1 = ? *cardrank2 = &CardCode2 = ?	card = 0 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = ' '	card = 0 *cardrank1 = &CardCode1 = ' ' *cardrank2 = &CardCode2 = ' '	**P**

#9 void identifyCardColor (int card, int* color)
Function description: stores the values 1 and 0 in ***color***
(1 = red / 0 = black)

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Ace of Clubs identifyCardColor (14, &Color);	card = 14 *color = &Color = ?	card = 14 *color = &Color = 0	card = 14 *color = &Color = 0	**P**
2	6 of Spades identifyCardColor (45, &Color);	card = 45 *color = &Color = ?	card = 45 *color = &Color = 0	card = 45 *color = &Color = 0	**P**
3	King of Hearts identifyCardColor (39, &Color);	card = 39 *color = &Color = ?	card = 39 *color = &Color = 1	card = 39 *color = &Color = 1	**P**
4	Queen of Diamonds identifyCardColor (12, &Color);	card = 12 *color = &Color = ?	card = 12 *color = &Color = 1	card = 12 *color = &Color = 1	**P**

#10 int is2Cards11 (int card1, int card2)
Function description: returns ***res*** -- 1 if 2 cards sums up to 11, returns 0 if not

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Number cards Card11 = is2Cards11 (10, 14);	card1 = 10 card2 = 14 Card11 = 0	card1 = 10 card2 = 14 Card11 = 1	card1 = 10 card2 = 14 Card11 = 1	**P**
2	Number cards Card11 = is2Cards11 (41, 19);	card1 = 41 card2 = 19 Card11 = 0	card1 = 41 card2 = 19 Card11 = 0	card1 = 41 card2 = 19 Card11 = 0	**P**
3	Same number cards Card11 = is2Cards11 (6, 32);	card1 = 6 card2 = 32 Card11 = 0	card1 = 6 card2 = 32 Card11 = 0	card1 = 6 card2 = 32 Card11 = 0	**P**
4	Royal cards Card11 = is2Cards11 (13, 11);	card1 = 13 card2 = 11 Card11 = 0	card1 = 13 card2 = 11 Card11 = 0	card1 = 13 card2 = 11 Card11 = 1	**F**

To fix #4, before calling the function, there are conditional statements to check if the cards are number, royal, or blank

#11 int isRoyalCard (int card)

Function description: returns ***res*** -- 1 if card is a royal (either Jack, Queen, or King)
calls isCardJack(), isCardQueen(), and isCardKing()

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Jack Royal = isRoyalCard (11);	isCardJack(11) || isCardQueen(11) || isCardKing(11) Royal = 0	Royal = 1	Royal = 1	**P**
2	Queen Royal = isRoyalCard (38);	isCardJack(38) || isCardQueen(38) || isCardKing(38) Royal = 0	Royal = 1	Royal = 1	**P**
3	King Royal = isRoyalCard (52);	isCardJack(52) || isCardQueen(52) || isCardKing(52) Royal = 0	Royal = 1	Royal = 1	**P**
4	Ace Royal = isRoyalCard (14);	isCardJack(14) || isCardQueen(14) || isCardKing(14) Royal = 0	Royal = 0	Royal = 0	**P**
5	0 Royal = isRoyalCard (0);	isCardJack(0) || isCardQueen(0) || isCardKing(0) Royal = 0	Royal = 0	Royal = 0	**P**

#12 int is3CardsRoyal (int card1, int card2, int card3)
Function description: returns ***res*** -- 1 if 3 cards are a set of Royals, 0 if not

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	All royal cards CardRoyal = is3CardsRoyal (11, 25, 39);	card1 = 11 card2 = 25 card3 = 29 CardRoyal = 0	card1 = 11 card2 = 25 card3 = 29 CardRoyal = 1	card1 = 11 card2 = 25 card3 = 29 CardRoyal = 1	**P**

2	Combination of royal and number cards CardRoyal = is3CardsRoyal (11, 21, 52);	card1 = 11 card2 = 21 card3 = 52 CardRoyal = 0	card1 = 11 card2 = 21 card3 = 52 CardRoyal = 0	card1 = 11 card2 = 21 card3 = 52 CardRoyal = 0	**P**
3	Some royal cards repeat CardRoyal = is3CardsRoyal (12, 38, 50);	card1 = 12 card2 = 38 card3 = 50 CardRoyal = 0	card1 = 12 card2 = 38 card3 = 50 CardRoyal = 0	card1 = 12 card2 = 38 card3 = 50 CardRoyal = 0	**P**
5	Rank Cards CardRoyal = is3CardsRoyal (2, 3, 5);	card1 = 2 card2 = 3 card3 = 5 CardRoyal = 0	card1 = 2 card2 = 3 card3 = 5 CardRoyal = 0	card1 = 2 card2 = 3 card3 = 5 CardRoyal = 1	**F**

To fix #5, before calling the function, there are conditional statements to check if the cards are number, royal, or blank

#13 int getScore11 (int card1, int card2)
Function description: returns ***score*** -- 3 if same suit, 2 if same color, 1 by default
calls identifyCardSuit() and identifyCardColor()

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Same suit Score = getScore11 (15, 22);	card1 = 15 card2 = 22 Score = 1	card1 = 15 card2 = 22 Score = 3	card1 = 15 card2 = 22 Score = 3	**P**
2	Same color Score = getScore11 (18, 45);	card1 = 18 card2 = 45 Score = 1	card1 = 18 card2 = 45 Score = 2	card1 = 18 card2 = 45 Score = 2	**P**
3	Different Score = getScore11 (7, 17);	card1 = 7 card2 = 17 Score = 1	card1 = 7 card2 = 17 Score = 1	card1 = 7 card2 = 17 Score = 1	**P**

#14 int getScoreRoyal (int card1, int card2, int card3)
Function description: returns ***score*** -- 3 if same suit, 2 if same color, 1 by default

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	Same suit Score = getScoreRoyal (24, 25, 26);	card1 = 24 card2 = 25 card3 = 26 Score = 1	card1 = 24 card2 = 25 card3 = 26 Score = 3	card1 = 24 card2 = 25 card3 = 26 Score = 3	**P**

2	Same color Score = getScoreRoyal (11, 38, 39);	card1 = 11 card2 = 38 card3 = 39 Score = 1	card1 = 11 card2 = 38 card3 = 39 Score = 2	card1 = 11 card2 = 38 card3 = 39 Score = 2	**P**
3	Different Score = getScoreRoyal (50, 38, 13);	card1 = 50 card2 = 38 card3 = 13 Score = 1	card1 = 50 card2 = 38 card3 = 13 Score = 1	card1 = 50 card2 = 38 card3 = 13 Score = 1	**P**
4	Different Score = getScoreRoyal (39, 38, 50);	card1 = 39 card2 = 38 card3 = 50 Score = 1	card1 = 39 card2 = 38 card3 = 50 Score = 1	card1 = 39 card2 = 38 card3 = 50 Score = 1	**P**

#15 int areRoyalsPresent (int jack, int queen, int king)
Function description: returns ***res*** -- 1 if a set of royal cards is present, returns 0 if not
results of isCardJack(), isCardQueen(), and isCardKing()

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	All present RPresent = areRoyalsPresent (1, 1, 1)	jack = 1 queen = 1 king = 1 RPresent = 0	jack = 1 queen = 1 king = 1 RPresent = 1	jack = 1 queen = 1 king = 1 RPresent = 1	**P**
2	Only one present RPresent = areRoyalsPresent (0, 1, 0)	jack = 0 queen = 1 king = 0 RPresent = 0	jack = 0 queen = 1 king = 0 RPresent = 0	jack = 0 queen = 1 king = 0 RPresent = 0	**P**
3	None present RPresent = areRoyalsPresent (0, 0, 0)	jack = 0 queen = 0 king = 0 RPresent = 0	jack = 0 queen = 0 king = 0 RPresent = 0	jack = 0 queen = 0 king = 0 RPresent = 0	**P**

#16 char choosePlayerMode()
Function description: stores the user player mode selection in ****pMode***

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	user input is = 1 choosePlayerMode (&PlayerMode);	tempMode = *pMode = &PlayerMode = 1	tempMode = *pMode = &PlayerMode = 1	tempMode = *pMode = &PlayerMode = 1	**P**

2	user input is = 2 choosePlayerMode (&PlayerMode);	tempMode = *pMode = &PlayerMode = 2	tempMode = *pMode = &PlayerMode = 2	tempMode = *pMode = &PlayerMode = 2	**P**
3	user input is > 2 choosePlayerMode (&PlayerMode);	tempMode = *pMode = &PlayerMode = 4	tempMode = *pMode = &PlayerMode = 4 *Display:* INVALID, back to 1	tempMode = *pMode = &PlayerMode = 4 *Display:* INVALID, back to 1	**P**
4	user input is < 1 choosePlayerMode (&PlayerMode);	tempMode = *pMode = &PlayerMode = -2	tempMode = *pMode = &PlayerMode = -2 *Display:* INVALID, back to 1	tempMode = *pMode = &PlayerMode = -2 *Display:* INVALID, back to 1	**P**

#17 char chooseGameMode()
Function description: stores the user game mode selection in ***gMode***

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	user input is = 1 (COMPLETE DECK) chooseGameMode (&GameMode);	tempMode = *gMode = &GameMode = 1	tempMode = *gMode = &GameMode = 1	tempMode = *gMode = &GameMode = 1	**P**
2	user input is = 2 (RANK CARDS) chooseGameMode (&GameMode);	tempMode = *gMode = &GameMode = 2	tempMode = *gMode = &GameMode = 2	tempMode = *gMode = &GameMode = 2	**P**
3	user input is > 2 chooseGameMode (&GameMode);	tempMode = *gMode = &GameMode = 4	tempMode = *gMode = &GameMode = 4 *Display:* INVALID, back to 1	tempMode = *gMode = &GameMode = 4 *Display:* INVALID, back to 1	**P**
4	user input is < 1 chooseGameMode (&GameMode);	tempMode = *gMode = &GameMode = -2	tempMode = *gMode = &GameMode = -2 *Display:* INVALID, back to 1	tempMode = *gMode = &GameMode = -2 *Display:* INVALID, back to 1	**P**

minor functions () test cases

#18 char chooseMenu()
Function description: this function asks for the player's selection in the menu and returns it -- ***selection***;
it doesn't validate anything so any input will be accepted
refer to **#1 Menu** in **main()** test cases for input validation

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	user input is N (new game) MenuSelection = chooseMenu();	-	MenuSelection = 'N'	MenuSelection = 'N'	**P**

2	user input is n (new game) MenuSelection = chooseMenu();	-	MenuSelection = 'n'	MenuSelection = 'n'	**P**
3	user input is S (settings) MenuSelection = chooseMenu();	-	MenuSelection = 's'	MenuSelection = 's'	**P**
4	user input is s (settings) MenuSelection = chooseMenu();	-	MenuSelection = 's'	MenuSelection = 's'	**P**
5	user input is E (exit) MenuSelection = chooseMenu();	-	MenuSelection = 'E'	MenuSelection = 'E'	**P**
6	user input is e (exit) MenuSelection = chooseMenu();	-	MenuSelection = 'e'	MenuSelection = 'e'	**P**
7	other inputs MenuSelection = chooseMenu();	-	MenuSelection = 'g' MenuSelection = 'F' MenuSelection = 'v'	MenuSelection = 'g' MenuSelection = 'F' MenuSelection = 'v'	**P**

#19 char chooseSettings()
Function description: this function asks for the player's selection in the settings and returns it -- ***selection***;
it doesn't validate anything so any input will be accepted
refer to **#2 Settings** in **main()** test cases for input validation

#	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	user input is P (player mode) SettingSelection = chooseSetting();	-	SettingSelection = 'P'	SettingSelection = 'P'	**P**
2	user input is p (player mode) SettingSelection = chooseSetting();	-	SettingSelection = 'p'	SettingSelection = 'p'	**P**
3	user input is G (game mode) SettingSelection = chooseSetting();	-	SettingSelection = 'G'	SettingSelection = 'G'	**P**
4	user input is g (player mode) SettingSelection = chooseSetting();	-	SettingSelection = 'g'	SettingSelection = 'g'	**P**
5	user input is B (back) SettingSelection = chooseSetting();	-	SettingSelection = 'B'	SettingSelection = 'B'	**P**
6	user input is b (back) SettingSelection = chooseSetting();	-	SettingSelection = 'b'	SettingSelection = 'b'	**P**
7	other inputs SettingSelection = chooseSetting();	-	MenuSelection = 'f' MenuSelection = 'T'	MenuSelection = 'f' MenuSelection = 'T'	**P**

main () test cases

#1 Menu
Part description: first part of running the program
calls chooseMenu()

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	User input is 'N' or 'n' (New Game)	MenuSelection = 'N' or MenuSelection = 'n'	enter new game	enter new game	**P**
2	User input is 'S' or 's' (Settings)	MenuSelection = 'S' or MenuSelection = 's'	enter settings	enter settings	**P**
3	Other input	MenuSelection = 'v' or MenuSelection = 'F' or MenuSelection = 'g'	*Display:* “Invalid Selection” ask for another input	*Display:* “Invalid Selection” ask for another input	**P**

#2 Settings
Part description: the settings part if chosen during Menu part
calls chooseSettings()

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	User input is 'P' or 'p' (Player Mode)	SettingSelection = 'P' or SettingSelection = 'p'	enter player mode selection	enter player mode selection	**P**
2	User input is 'G' or 'g' (Game Mode)	SettingSelection = 'G' or SettingSelection = 'g'	enter game mode selection	enter game mode selection	**P**
3	User input is 'B' or 'b' (Back to Main Menu)	SettingSelection = 'B' or SettingSelection = 'b'	go back to menu	go back to menu	**P**
4	Other input	SettingSelection = 'f' or SettingSelection = 'T'	*Display:* “Invalid Selection” ask for another input	*Display:* “Invalid Selection” ask for another input	**P**

#3 RCard or NCard
Part description: the user is asked if s/he is to eliminate royal card or number card
this is also the part where player is given a chance to go back to menu

	Test Description	Sample Input	Expected Result	Actual Result	P/F

1	User input is 'R' or 'r' (Royal Card)	RCardNCard = 'R' or RCardNCard = 'r'	proceeds to royal card selection	proceeds to royal card selection	**P**
2	User input is 'N' or 'n' (Number Card)	RCardNCard = 'n' or RCardNCard = 'N'	proceeds to number card selection	proceeds to number card selection	**P**
3	Other input	RCardNCard = 'e' or RCardNCard = 'T'	*Display:* “Invalid Selection” ask for another input	*Display:* “Invalid Selection” ask for another input	**P**

#4 CardSelect
Part description: the user is asked to select cards

	Test Description	Sample Input	Expected Result	Actual Result	P/F
1	0 < Card < 10	Card = 8	proceeds to next part	proceeds to next part	**P**
2	0 < Card < 10	Card = 3	proceeds to next part	proceeds to next part	**P**
3	Card = 0 (during first card selection)	Card = 0	go back to menu	go back to menu	
4	Card = 0 (during second/third card selection)	Card = 0	*Display:* “Wrong Cell” ask for another input	*Display:* “Wrong Cell” ask for another input	**P**
5	Card < 0	Card = -8	*Display:* “Wrong Cell” ask for another input	*Display:* “Wrong Cell” ask for another input	**P**
6	Card > 9	Card = 13	*Display:* “Wrong Cell” ask for another input	*Display:* “Wrong Cell” ask for another input	**P**

Additional Documentation:

- For cards that are already selected/used, they are replaced with 0 in the array. Hence, some functions accept 0 as input.
- For the shuffling of cards, the Fisher-Yates algorithm is used, and the tutorial by Portfolio Courses on YouTube was used as a guide.
- To learn about how to use rand(), srand(), and time.h, I followed LearningLad on YouTube.

Program Sequence:

A. Menu

a. New Game → proceeds to ***B.***
b. Settings
   i. Player Mode
   ii. Game Mode
c. Exit → ends the program

B. Start of Game

a. Initialize
   i. Filling the card array
   ii. Shuffling the card array using the Fisher-Yates algorithm
   iii. Setting the first 9 cards

   BONUS: Checking if the card is a Royal. If it is, another card is selected.
b. Printing the 9 cards
c. Printing player status and scores
   - Player Mode 2:
     - First player: Player 1
     - Default scores: 0
d. Checking if the table is playable
   i. Checking if there are two cards in the table that sum up to 11
   ii. Checking if there is a complete set of royal cards
   - If the table is not playable (meaning no elevens and no royal set):
     - Player Mode 1: printing of “win” and “game over” messages
     - Player Mode 2: Printing of the winner
       - In case of a tie, “tie” message is printed
e. Card selection
   i. Royal Card or Number Card (executes only when game mode is 1)
   ii. 1st card
   iii. 2nd card
   iv. 3rd card (executes only when ‘R’ is selected)
f. Matching the cards with user selection
g. Validating if cards are Elevens or Royals
h. Replacing cards (only executes if ***g.*** is valid)

BONUS: Checking if the card is a Royal. If it is, another card is selected.

i. Return to ***b.***