Lab 7: Reading Input and Input Redirection

**Due date:** Wednesady, February 1, 11:59pm.

# Lab Assignment

## Assignment Preparation

This is an individual assignment.

**Collaboration.** You can talk to each other during the lab, but no code sharing is allowed. You may not speak to anyone outside the class.

**Purpose.** This lab allows you to practice the use of `scanf()` function to read input from the keyboard. You will also get a chance to use input redirection from file for this lab, and to use out parameters (a.k.a., call by reference) in functions.

**Programming Style.** All submitted C programs must adhere to the programming style described in detail at

http://users.csc.calpoly.edu/∼cstaley/General/CStyle.htm

When graded, the programs will be checked for style. Any stylistic violations are subject to a 10% penalty. Significant stylistic violations, epsecially those that make grading harder, may yield stricter penalties. Also note the the Lab 2 requirement for the content of the header comment in each file you submit applies **to each assignment** (lab, programming assignment, homework) in this course.

**Testing and Submissions.** Any submission that does not compile using the

```
gcc -ansi -Wall -Werror -lm
```

compiler settings will receive an automatic score of 0.

## The Task

### Program 1: Evaluation of Blackjack hands. `blackjack.c`

Your first program shall ask the user to input information about up to four cards. It will then compute the value of the cards the user entered using the rules of the casino blackjack.

**BR-0.** Name your program `blackjack.c`. Your program shall contain an `int main()` function and a number of other functions, described below. All the functions shall be declared and defined in the `blackjack.c` file.

**BR-1. Overview.** The work of the program will proceed as follows. When started, the program prints a prompt and asks the user to enter the cards the user has. Up to four cards can be entered. The user can enter less then four cards, but providing "special" inputs to the program indicating that no more cards are to be read.

After the input cards are read, their value is computed according to the rules of Casino Blackjack. The rules are specified below. The *favored* value of the hand is printed to the user.

**BR-2. Blackjack.** The game of Casino Blackjack is played with a multiples of the traditional 52-card four suit deck. Usually, a **Casino Blackjack** deck contains around 6-10 52-card decks. This means that the card can show up multiple times in a single hand dealt to a player. Therefore, no hand validity checks shall be conducted.

The card decks have four suits, but for the game of Casino Blackjack the suits are not relevant, so we will ignore them. Cards come in denominations: 2,3,4,5,6,7,8,9,10, Jack, Queen, King and Ace.

Each card in Casino Blackjack has a numeric value. All numeric cards have the value indicated by the card, i.e., the value of a 2 is 2, the value of a 5 is 5, and so on. Jack, Queen and King all have the value of 10.

Aces are a special case. An Ace can have a value of either 1 or 11 depending on what suits the player better.

The *value of a hand* is the sum of the values of all cards in the hand.

A Blackjack is a combination of an Ace and any card that has a value of 10. The numeric value of a Blackjack is 21. Note, that there are other ways to achieve 21 in the game (e.g., a four card hand of 5, 6, 3 and 7), but only a two-card hand of an Ace and a 10-valued card is a proper blackjack.

A hand with a total value exceeding 21 is called a bust. A player bust loses the round, so blackjack players must avoid busts.

A group of hands that involve Aces may become busts if the value of an Ace is counted as 11, but may have values below 21 if the value of an Ace is counted as 1. For example a hand of Ace, 6, 8 would be a bust: $11 + 6 + 8 = 25$ if the Ace is counted as 11, but it's value drops to $1 + 6 + 8 = 15$ when the Ace is counted as 1.

In Casino Blackjack, Ace value automatically becomes the one that benefits the player the most. If the two hand values are $X$ and $Y$, and $X < Y \leq 21$, then we select the value $Y$ — the larger of the two (but not a bust). If $X \leq 21 < Y$, the we select $X$, as the value that does not lose the round immediately.

Hands with two or more Aces are handled as follows. The first Ace can be counted as either having a value of 1 or a value of 11. All following Aces must be counted as having the value of 1. Note, that two Aces counted as 11 is a bust score, so *at most one Ace per hand can be counted as* 11.

**BR-3. Start of the program.** When the program starts, it prints the following text (make sure to match it exactly):

```
=== Blackjack Hand Evaluator, Version 1.0 ===
```

after which it skips one line.

**BR-4.** Your program shall read up to four card values as follows. For each card to be read, your program shall produce a prompt

```
Card #<N>:
```

(with a space following the `:` character). Here `<N>` is replaced with `1`, `2`, `3` or `4` based on which card value is ready.

Following the prompt your program shall read a single `int` value: the value of the next card. The following values shall be accepted by your program:

| Value accepted | Cards Represented |
|---|---|
| 2...9 | 2 through 9 respectively |
| 10 | 10, Jack, Queen, King |
| 1 | Ace |
| 0 | End of card entry. |

The behavior of your program on any other input remains undefined. No test cases will test such inputs.

The value 0 entered instead of a card value means that no more cards have left to supply.

**BR-5. Program behavior.** Upon reading the value of the next card, your program shall perform the following activities:

1. Check if the *end of card entry* signal is received (a value of 0 is entered).
2. If yes, *or if the fourth card is entered*, output the number of cards and their total value (see below).
3. If no, and only zero, one, two, or three cards have been entered, the program shall recompute the current hand value(s) and proceed to read the next card in the hand (see **BR-4**).

**BR-6. Update.** Each time a new card is entered, your program shall recompute the possible hand values. This is achieved in the following way:

- The program keeps and updates appropriately two *running totals* for the hand values. The first score represents the value of the hand, when *all*

*Aces are valued at* 1. The second score represents the value of the hand when *the first Ace in the hand is valued at* 11, while all other Aces are valued at 1. On hands with no Aces the values of the two scores must coincide.

- The program keeps and updates a counter of the number of cards in the hand.

**BR-7. Endgame.** When the fourth card is entered, or when a 0 is entered as the value of the next card in the hand, the program shall complete its operation. The program shall choose the crrent hand value from the two values maintained. This is done according to the rule specified in **BR-2**:

> If both possible hand totals are less than or equal to 21, *pick the larger of the two.* Otherwise, *pick the smaller of the two.*

Please note, that it is possible for both values of the hand to be bust. E.g., the hand 10,8,9,A is a bust[1]

**If no cards** have been entered, your program shall print

```
You have no cards.
```

(notice the period at the end of the line). This text must end with a new line character.

**if at least one card** has been entered, your program shall print two lines of text:

```
You have <N> cards.
Hand value: <VAL>
```

Here, `<N>` shall be replaced with the number of cards in the user's hand, while `<VAL>` shall be replaced with the total computed value of the entered Blackjack hand.

Upon printing the value of the cards, your program shall perform two more checks. If the current card hand value exceeds 21 (i.e., is 22 or above), print (on a separate line):

```
Bust!
```

Additionally, if the current card hand is equal to 21, **and** it consists of an Ace and a 10, print (on a separate line):

```
Blackjack!
```

**BR-8. Functions.** Declare, define, and **use** in your code, the following functions:

```
void printGreeting();
```

[1] Of course, since 10,8,9 is already a bust, such a four-card hand is impossible in the actual game. However, for your program, you will compute this value faithfully and report is the same way as other values.

```
void getCard(int * card, int n);
void updateScores(int card, int *lowSum, int *highSum, int * cardCount,
                  int * seenAce);
void printComments(int hand, int count);
int getHandValue(int lowsum, int highsum);
```

`printGreeting()` shall print the initial greeting message of the program.

`getCard()` takes as input an integer `n`: the ordinal of the card to be read (1,2,3 or 4) and one out parameter. The function shall read the value of the card from input stream using `scanf()`.

`updateScores()` takes as input the currently provided card value, and four out parameters:

| Parameter | Explanation |
|---|---|
| `lowSum` | the hand value count that treats all Aces as 1. |
| `highSum` | the hand value count that treats the first Ace as 11. |
| `cardCoubnt` | the total number of cards in the hand *prior* to reading the current card value |
| `seenAce` | and integer flag specifying where an Ace has already been seen in the hand. |

The function shall update the four values above based on the value of the new card entered.

`printComments()` is a function that determine if the current (completed) hand is a bust or if it is a blackjack. The function shall print the required text (see **BR-7**).

`getHandValue()` takes as input the two possible value scores for the card hands (as maintained by the `updateScores()` function. It chooses the score of the hand based on the relationship between these scores and the value 21 as described in **BR-2**.

**Please note,** that `getCard()` and `updateScores()` have *out* parameters. Be careful when writing assignment statements that update these parameters - *use the correct syntax.*

**Another note.** The filename is `blackjack.c`. If you compile this file into a `blackjack` program you should be aware that there may be a `blackjack` program in the `/usr/bin` directory. Because of this, either always run your program using

```
> ./blackjack
```

(stressing that you want to run the executable from the current directory), or (which is easier), compile into a program with a different name.

## Program 2:Electoral College results. `elections.c`

In prior versions of this course, variants of this program were offered early in the course to (a) allow students to write a long-ish program during the second week of classes and to (b) later keep on improving on it.

This program is rather large, but repetetive — feel free to use copy-and-paste as it suits you.

In 2008, the United States held a Presidential election. There were two major party candidates running, Barak Obama (Democratic Party) and John McCain (Republican Party). The electoral system of the United States, which you all have studied in high school is unique in its use of the electoral college system. Each US state is awarded a number of votes in the electoral college equal to the total number of its Congressional delegation (two Senators plus the number of Representatives; District of Columbia gets 3 electoral votes). The popular vote winner in the state receives the state's electoral college votes. Nebraska and Maine award their electoral votes slightly differently: by giving two electoral college votes to the winner of the popular vote in the state, and giving the remaining votes individually to winners in each of the Congressional districts (Maine has two Congressional districts, Nebraska has three). There is a total of 538 electoral votes at stake, and 270 votes win the election. The 269-269 tie is broken in the House of Representatives.

In the 2008 Presidential election Barak Obama has won 365 electoral votes to John McCain's 173 to become the President of the U.S.

You will write a program that will compute the electoral college votes received by the two major candidates for President in a Presidential election. The input to the program will be a sequence of 51 numbers indicating who won the popular vote in each state (and the District of Columbia). The output of the program will be the total number of electoral college votes each candidate received and the information about who won the elections.

## Non-Functional Requirements

This is going to be a rather naïve and tedious-to-implement program, but correct implementation will help you understand the need for C constructs we will be studying in the next few weeks.

**ECN1.** The table of Electoral College votes is found in **Appendix A.**.

**ECN2.** You shall use 51 C constants (using `#define` directive) to represent the electoral college votes of each state/DC. Each constant shall be named after the two-letter state abbreviation: AL, AK, AZ, CA, CT, DC, GA, FL, etc.... The value of each constant comes from the table mentioned in **ECN1.**.

**ECN3.** You shall also use one more C constant, `ELECTORALCOLLEGE`, whose value is set to 538 — the total number of votes in the Electoral College.

**ECN4.** Your program shall be named `elections.c`.

## Functional Requirements

**ECF0.** The program shall read from the input stream 51 numbers representing the election results in each of 50 US states and DC. Along the way, it will keep track of the number of electoral college votes for one of the major party candidates and determine who won each state. After the election results from all 50 states and DC are read in, the program will obtain the electoral college vote total for each candidate and will output it. It will also determine the winner of the elections and output the winner's name.

The input to the program will come from a file via input redirection. Because of this, there is no need to produce any *input prompts* using `printf()` functions calls. You will only use `printf()` calls to print the results of your computations.

The entire code shall reside in the `int main()` function of the `elections.c` file. No other functions shall be used.

**ECF1.** The input to the program is a sequence of 51 zeroes and ones. Each number in a sequence represents the results of the popular vote in one of the states or DC. The vote results come in order of appearance of states in the **Appendix A** table (alphabetical order by full state name). That is, first number in the sequence specifies the results of popular vote in Alabama, second — in Alaska, third – in Arizona, and so forth.

The input will mean the following:

- **0**: popular vote in the state is **won by John McCain**.
- **1**: popular vote in the state is **won by Barack Obama**.

**The only exception to this rule is the input number representing the voting results in Nebraska.** This is the input number 28 to this program (See Appendix A). This number **is equal to the total number of electoral votes Barack Obama received from the State of Nebraska.** This number is guaranteed to be in the range between 0 and 5[2]

Example. Suppose the first seven numbers in the input sequence are `0 0 0 0 1 1 1`. This would mean (check the Electoral College table in **Appendix A**) that John McCain has won the popular vote in Alabama, Alaska, Arizona and Arkansas, while Barack Obama has one the popular vote in California, Colorado and Connecticut[3].

**ECF2.** Your program shall use two integer variables to keep track of the electoral college vote for each candidate. While you have the option of naming them differently, in the rest of the requirements we will refer to them as `votesMcCain` and `votesObama`.

Your program also shall contain a single integer variable, we refer to as `stateDecision` to represent the input information.

**ECF3.** Both `votesMcCain` and `votesObama` are initialized to 0 at the beginning of the program.

**ECF4.** The program repeats the following sequence of actions 51 times (for each input number read):

**ECF4-1.** The program reads the next input number into the `stateDecision` variable.

**ECF4-2.** The program updates the popular vote tally for Barack Obama. This is done by computing the number of electoral college votes Barack Obama received from the state whose decision has just been read from the input

[2]In the actual election, Obama won Nebraska's Congressional district 2 centered around Omaha, NE and received one Electoral College vote from this state.

[3]Which is what they actually did in 2008

stream and adding this number to the current popular vote tally for Barack Obama.

Notice that for all states but Nebraska, stateDecision =1 if Barack Obama won the state, and 0 if he lost it. Thus, the number of electoral college votes Barack Obama receives from the state can be computed as `stateDecision * <ST>` where `<ST>` is the constant representing the electoral college vote of the state in question (i.e., `AL` for the first input number, `AK` for the second, etc...). In case of Nebraska, the number of votes Barack Obama won will be read into the `stateDecision` variable directly (this is the easy case!).

**ECF4-3.** The program determines who won the electoral college vote in the state. If the winner is Barak Obama, the program shall output the following text:

```
<STATE> goes to Obama
```

If the winner is John McCain, the program shall output the following text:

```
<STATE> goes to McCain
```

Here, `<STATE>` is the name of the state under consideration. For example, if the first seven numbers in the input are `0,0,0,0,1,1,1`, the first seven lines of output produced by your program will be:

```
Alabama goes to McCain
Alaska goes to McCain
Arizona goes to McCain
Arkansas goes to McCain
California goes to Obama
Colorado goes to Obama
Connecticut goes to Obama
```

**Note: ECF4** actually means that you have to **repeat** roughly the same piece of code 51 times (this is where copy-and-paste help). *Even if you DO know how to use loops in C, **don't use them here** - you will be penalized for it - we will redo this program using loops and arrays later in the course.*

**ECF5.** After the actions described in **ECF4** have been performed 51 times, your `votesObama` variable will contain the number of votes in the entire electoral college received by Barack Obama. Next, your program shall compute the number of electoral college votes received by John McCain. To do this, subtract the number of the electoral college votes Barack Obama received from the total number of the electoral college votes (represented as the `ELECTORALCOLLEGE` constant in your program).

**ECF6.** Your program shall output the electoral college votes for both candidates. The following two lines shall be printed:

```
Electoral College vote total for John McCain (R) is <votesMcCain>
Electoral College vote total for Barack Obama (D) is <votesObama>
```

Here, your program outpus the values of variables `votesMcCain` and `votesObama` where the placeholders `<votesMcCain>` and `<votesObama>` are used above. After this, the program ends its work.

**ECF7.** The final line of the output shall announce the winner of the elections. If Barak Obama has more electoral college votes than John McCain, output

```
Barak Obama won the election!
```

If John McCains has more electoral college votes than John McCain, output

```
John McCain won the election!
```

If the electoral vote is a 269-to-269 tie, then output

```
It's a tie!
```

**Note:** Instructor's executable `elections-alex` is made available to you, as well as a collection of tests. The output of your program and the output of the instructor's program **must coincide**. There are **no spaces** at the end of any lines in the instructor's program. Any differences in the output are subject to penalties.

## Submission.

**Files to submit.** You shall submit two files, `blackjack.c` and `elections.c`.

Use the `handin` command.

```
> handin dekhtyar lab07 blackjack.c elections.c
```

**Please, DO NOT submit binary files**.

## Appendix A. Electoral College

| No. | State | Abbr. | Population | Electoral College Votes |
|---|---|---|---|---|
| 1. | Alabama | AL | 4,500,752 | **9** |
| 2. | Alaska | AK | 648,818 | **3** |
| 3. | Arizona | AZ | 5,580,811 | **10** |
| 4. | Arkansas | AR | 2,725,714 | **6** |
| 5. | California | CA | 35,484,453 | **55** |
| 6. | Colorado | CO | 4,550,688 | **9** |
| 7. | Connecticut | CT | 3,483,372 | **7** |
| 8. | Delaware | DE | 817,491 | **3** |
| 9. | District of Columbia | DC | 563,384 | **3** |
| 10. | Florida | FL | 17,019,068 | **27** |
| 11. | Georgia | GA | 8,684,715 | **15** |
| 12. | Hawaii | HI | 1,257,608 | **4** |
| 13. | Idaho | ID | 1,366,332 | **4** |
| 14. | Illinois | IL | 12,653,544 | **21** |
| 15. | Indiana | IN | 6,195,643 | **11** |
| 16. | Iowa | IA | 2,944,062 | **7** |
| 17. | Kansas | KS | 2,723,507 | **6** |
| 18. | Kentucky | KY | 4,117,827 | **8** |
| 19. | Louisiana | LA | 4,496,334 | **9** |
| 20. | Maine | ME | 1,305,728 | **4** |
| 21. | Maryland | MD | 5,508,909 | **10** |
| 22. | Massachusetts | MA | 6,433,422 | **12** |
| 23. | Michigan | MI | 10,079,985 | **17** |
| 24. | Minnesota | MN | 5,059,375 | **10** |
| 25. | Mississippi | MS | 2,881,281 | **6** |
| 26. | Missouri | MO | 5,704,484 | **11** |
| 27. | Montana | MT | 917,621 | **3** |
| 28. | Nebraska | NE | 1,739,291 | **5** |
| 29. | Nevada | NV | 2,241,154 | **5** |
| 30. | New Hampshire | NH | 1,287,687 | **4** |
| 31. | New Jersey | NJ | 8,638,396 | **15** |
| 32. | New Mexico | NM | 1,874,614 | **5** |
| 33. | New York | NY | 19,190,115 | **31** |
| 34. | North Carolina | NC | 8,407,248 | **15** |
| 35. | North Dakota | ND | 633,837 | **3** |
| 36. | Ohio | OH | 11,435,798 | **20** |
| 37. | Oklahoma | OK | 3,511,532 | **7** |
| 38. | Oregon | OR | 3,559,596 | **7** |
| 39. | Pennsylvania | PA | 12,365,455 | **21** |
| 40. | Rhode Island | RI | 1,076,164 | **4** |
| 41. | South Carolina | SC | 4,147,152 | **8** |
| 42. | South Dakota | SD | 764,309 | **3** |
| 43. | Tennessee | TN | 5,841,748 | **11** |
| 44. | Texas | TX | 22,118,509 | **34** |
| 45. | Utah | UT | 2,351,467 | **5** |
| 46. | Vermont | VT | 619,107 | **3** |
| 47. | Virginia | VA | 7,386,330 | **13** |
| 48. | Washington | WA | 6,131,445 | **11** |
| 49. | West Virginia | WV | 1,810,354 | **5** |
| 50. | Wisconsin | WI | 5,472,299 | **10** |
| 51. | Wyoming | WY | 501,242 | **3** |