APS 105 — Computer Fundamentals

Lab #3: Simple Blackjack
Winter 2021

Important note: You must use the `submit` command to electronically submit your solution by the end of your lab day.

In this lab, you will write a C program / application that plays a simple game of Blackjack. By the end of this lab, you should be able to:

- Accept and validate user input from the keyboard.
- Use the user input to perform and output useful calculations.
- Make decisions based on different conditions.
- Loop (i.e., repeat code) a certain number of times based on different conditions.
- Explicitly format the output of numbers.

1 Problem Statement

In this simplified version of Blackjack, we assume the following:

- The "cards" are random numbers from 1 to 13. We will allow repeats without limit of any of these numbers in this simplified game.
- At the start the "player" (the one using the app) and the "dealer" get one card each. These cards are known to the player.
- Cards are worth what the number is, except that 10-13 are all worth 10. So a 1 card is worth 1, a 2 card is worth 2, ..., a 9 card is worth 9, a 10 card is worth 10, an 11 card is worth 10, a 12 card is worth 10 and a 13 card is worth 10.
- The player can type '1' or '0' for hit or stay. A hit means they will get another random card (that they see). A stay means they are finished getting cards. They continue being asked until they stay, or until they lose because they are over 21. The player loses if the total worth of their cards exceeds 21.
- If the player stays with a total worth of 21 or under, then the dealer gets random cards until the dealer total is 17 or more.
- If the player gets over 21, or the dealer gets more than the player without going over 21, then the dealer wins. If the dealer and the player get the same value 21 or less, it is a tie. If the player stays with 21 or less and the dealer goes over 21 or if the dealer has less than the player, the player wins.

Note that playing with a real card deck and with a full set of Blackjack rules is a different game!

Examples (N means dealer doesn't play). Remember that cards from 10-13 are worth 10; totals are in square braces:

Player	Dealer	Winner	Note
3+1+9+11[23]	N	dealer	player over 21
12+11 [20]	6+6+13 [22]	player	dealer over 21
12+11 [20]	12+8 [18]	player	dealer must stop with 17 or over
5+3+10 [18]	1+8+9 [18]	tie	same total

2 Examples

Here are examples of how the game should interact with the player. Note that the dealer total is comprised of the initial card plus the new cards drawn after the Player is done. The information in angle brackets is not generated - it is an explanation of what is shown in the sample output:

<Dealer goes over 21>

```
First cards: player 12, dealer 6
Type 1 for Hit, 0 to Stay:1
Player gets a 3, worth is 13
Type 1 for Hit, 0 to Stay:1
Player gets a 7, worth is 20
Type 1 for Hit, 0 to Stay:0
Dealer gets: 6 13
Dealer over 21, Player wins
```

<Player and Dealer get the same worth>

```
First cards: player 1, dealer 5
Type 1 for Hit, 0 to Stay:1
Player gets a 8, worth is 9
Type 1 for Hit, 0 to Stay:1
Player gets a 10, worth is 19
Type 1 for Hit, 0 to Stay:0
Dealer gets: 4 11
Tie!
```

<Player goes over 21>

```
First cards: player 5, dealer 3
Type 1 for Hit, 0 to Stay:1
Player gets a 1, worth is 6
Type 1 for Hit, 0 to Stay:1
Player gets a 6, worth is 12
Type 1 for Hit, 0 to Stay:1
Player gets a 10, worth is 22
Player over 21, Dealer wins
```

<Both under 21, Dealer better hand>

```
First cards: player 6, dealer 12
Type 1 for Hit, 0 to Stay:1
```

```
Player gets a 12, worth is 16
Type 1 for Hit, 0 to Stay:0
Dealer gets: 8
Dealer better than Player, Dealer wins
```

«Note: similar if Player gets better hand, but with appropriate changes in final response»

3 Valid Input

You must ensure that each input from the player (1 or 0) is valid before proceeding with the rest of the program. Repeat the prompt for user input if not.

4 Notes

1. Your submitted code must NOT play multiple games, but you may want to put in a simple loop when you debug / do final verification checks. Be sure to take this out before you submit your code!!!
2. Your program must generate "random" cards with each run. But you want to be able to check your code with the same response every time on multiple samples, and we need to be able to check your code during the automarking. Therefore IMPORTANT:
   - Include the following lines of code in your program before any debugging loop:

     ```
     if(argc==1)
       srand(time(NULL));
           else
       srand(atoi(argv[1]));
     ```

     (You will get an explanation of the argc later. It relates to running your program through a command line - for example you could then use the command line **yourProgramName 16** to execute the program with the seed set to 16!)
   - Do NOT change the default

     ```
     int main(int argc, char **argv)
     ```

   - For debugging only, change the **time(NULL)** to a constant seed of your choice so you get consistent results. Be sure to change it back for submission purposes!!
3. You do not have to track individual cards dealt to player or dealer, but the output "Dealer gets:" should allow for multiple cards to be printed. For example:

   ```
   Dealer gets: 12
   Dealer gets: 5 3 2 2 13
   ```

4. Note that to generate a random number from 1-13 you should use

   ```
   rand() % 13 + 1;
   ```

   You must be able to explain why this is the code to use to your TA, and why each term of the calculation is there.

5. Suggestion: Make a chart of all the possible game outcomes, trace your code to make sure you've covered them, then test your code for each case.

6. This program has several stages of operation. To start it, first set up a list of comments to start a section of code for each stage or area of code. For example:

   ```
   \\variables define and initialize here
   \\ first card to player and dealer
   ...
   ```

   Next fill in each section. This keeps the amount of code to do under control - you aren't trying to do everything at once.

7. Learn the `<ctrl>c` and `<ctrl>v` keyboard commands for copy and paste. Much of the code for this program is the same or similar to other parts of the program. This copy/paste will make things faster!

5 Grading / Submission and Automarking

In a file called `Lab3.c`, write your solution to the problem. There are ten (10) marks available in this lab.

Read the notes and other comments carefully to make sure your automarked version conforms to expectations.

The total of **10 marks** on this lab are marked in two different ways:

(a) **By your TA, for 4 marks out of 10.** Once you are ready, show your program to your TA so that we can mark your program for style, and to ask you a few questions to test your understanding of what is happening.

   Programs with good style have been described in previous labs, but here is part of what might be looked for:

   - Good choices for variable names that indicate their purpose.
   - A consistent naming convention. Use camelCase for normal variable names.
   - Comments that explain code that is difficult to understand.
   - Proper indentation.
   - Appropriate white space between lines for better readability.

   The TA will also ask you some questions to be sure that you understand the underlying concepts being exercised in this lab.

(b) **By an auto-marking program for 6 marks out of 10**. You must submit all of your program files through the ECF computers for marking. Long before you submit your program for marking, you should run the **exercise** program that compiles and runs your program and gives it sample inputs, and checks that the outputs are correct. Similar to previous labs you should run the following command:

   ```
   /share/copy/aps105s/lab3/exercise
   ```

within the directory that contains your program.

This program will look for the file `Lab3.c` in your directory, compile it, and run it on a some of the test cases that will be used to mark your program automatically later. If there is anything wrong, the **exercise** program will report this to you, so read its output carefully, and fix the errors that it reports.

Once you have determined that your program is as correct as you can make it, then you must submit your program for auto-marking. This must be done by 11:59pm (Toronto time) the day of your lab period as that is the due time. To do so, go into the directory containing your program and type the following command:

```
/share/copy/aps105s/lab3/submit
```

This command will re-run the exercise program to check that everything looks fine. If it finds a problem, it will ask you if you are sure that you want to submit. Note that you may submit your work as many times as you want prior to the deadline; only the most recent submission is marked.

The **exercise** program (and the **marker** program that you will run after the final deadline) will be looking for the exact letters as described in the output in this handout, including the capitalization. When you test your program using the exercise program, you will see that it is expecting the output to be exactly this, so you will have to use it to see if you have this output correct.

Important Note: You must submit your lab by the designated time and date. Late submissions will not be accepted, and you will receive a grade of zero.

You can also check to see if what you think you have submitted is actually there, for peace of mind, using the following command:

```
/share/copy/aps105s/lab3/viewsubmitted
```

This command will download into the directory you run it in, a copy of all of the files that have been submitted. If you already have files of that same name in your directory, these files will be renamed with a number added to the end of the filename.

---

6 After the Final Deadline — Obtaining Automark

Briefly after all lab sections have finished (after the end of the week), you will be able to run the automarker to determine the automarked fraction of your grade on the code you have submitted. To do so, run the following command:

```
/share/copy/aps105s/lab3/marker
```

This command will compile and run your code, and test it with all of the test cases used to determine the automark grade. You will be able to see those test cases output and what went right or wrong.