COMP1511: Assignment 2
Card Game

The specification may change, so please check the change log on this page frequently.

ChangeLog

- [09:30am 25/Oct] : Latest stage2.c file is at stage2_v2.c, after minor typo fixed, see here.
- [11:10pm 24/Oct] : The supplied `*.o` files for "Testing Player.c" are updated, you can copy the latest versions by copying all *.o files to your "runner" dir using
  `% cp /home/cs1511/public_html/18s2/assigns/ass2/runner/*.o .`
- [10:11pm 24/Oct] : Additional debug printing added to stage1.c and stage2.c, updated files are now available (stage1_v2.c and stage2_v1.c). Please note that the tests are unchanged, only additional print statements are added to help in debugging.
- [10:35am 23/Oct] : Instructions on how to submit are now available, see **"Submitting"** below. **Deadline** is now **extended** to 11:59pm Friday 26/Oct.
- [02:00pm 21/Oct] : Instructions on **how to test player.c** updated: another new version of Game.o (with a few more bugs fixed) is available.
- [01:10pm 20/Oct] : Instructions on **how to test player.c** updated: a new version of Game.o (with a few bugs fixed) is available.
- [02:55pm 18/Oct] : Instructions on **how to test player.c** updated, the required files are now available.
- [01:50pm 17/Oct] : Instructions added on how to test your `player.c`. We are no longer going to run competitions, because they will not properly evaluate your `player.c`, to be discussed in the lecture.
- [06:55am 10/Oct] : Sample file for a player "player_sample.c" (as discussed in the lecture) is now available.
- [06:55am 10/Oct] : Sample testing file for "stage2" (as discussed in the lecture) is now available.
- [07:55am 09/Oct] : See the revised list of functions (as discussed in the lecture) not available to player.c, added "getHandCard", "playerCardCount" and "playerPoints". The following comment added (as discussed in the lecture) to the functions that return "Card"
  `// If no such card exists, the function returns NULL.`
- [07:45am 09/Oct] : Sample testing file for "stage1" is now available.
- [02:45pm 03/Oct] : Mark breakdown chanegd to Game.c (50%) and testGame.c (20%).
- [02:40pm 03/Oct] : Penalty of 3% of the marks awarded to you for this assignment will be applied for each missed submission (for weeks 10, 11 and 12).
- [12:18pm 02/Oct] : one argument (Game) added to the functions `getDeckCard, getDiscardPileCard and getHandCard` in Game.h.
- [07:30am 02/Oct] : Functions `getDeckCard, getDiscardPileCard and getHandCard` are added to Game.h, useful for testing (testGame.c).
- [02:40pm 24/Sep] : **FAQ** added
- [12:20pm 24/Sep] : Function `getActiveDrawTwos` is added to Game.h.
- [12:40pm 22/Sep] : Function `getTopDiscardTurnNumber` is added to Game.h .
- [06:30pm 21/Sep] : Clarifications for "first card in the discard pile at the start" added as list item 5 in the section "Playing the Game" .
- [11:30am 21/Sep] : I have added more information and rephrased some text to improve clarity in the section "Playing the Game".
- [11:30am 21/Sep] : The following functions are added to `Game.h`: `getNumberOfTwoCardsAtTop, getCurrentColor, getPreviousTurnPlayer`. Read comments in the file `Game.h` for more details on these three functions .

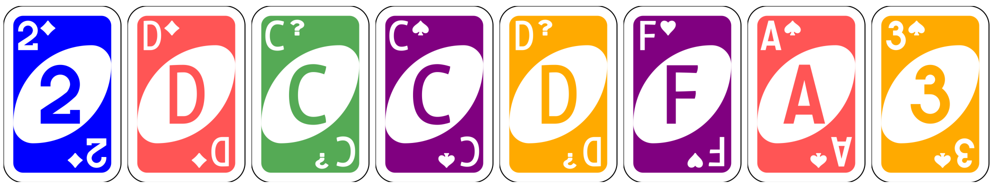
Goto: Objectives | Before you begin... | The Task | Competition | Submitting | Assessment | Plagiarism |

FAQ | **the Game Rules** | Sample testing for "stage1" | Sample testing for "stage2" | Sample file for a player "player_sample.c" |

The Card Game

The game *Final Card-Down* is a card game played with a special printed deck.

Each card has a *value* between 0 and F, a *suit* of either hearts, diamonds, clubs, spades, or questions, and a *color* of either red, green, blue, yellow, or purple.



Each *player* is dealt a *hand* of 7 cards. The first *player* to place their *final card down* in the discard pile wins.

For more details about the *game*, see the **Final Card-Down Rules**

Deadline (Individual assignment)

This is an **individual**.

This assignment is due on **Thursday, Week-13** , at **23:59:00** (Sydney time).

Objectives

For this assignment, not only will you be implementing a card game, but you'll also be writing a *player* (say using an *artificial intelligence*!) to play the game.

1. Implement the ADT (Game.h) and the rules of the card game.
2. Implement a set of *tests* (in testGame.c) for an *Abstract Data Type* (ADT) that represents the rules of a card game (Game.h).
3. Write a Player (say using an *artificial intelligence*) to play the game for you (player.c).

Before you begin…

Making a plan

Make a plan about how to approach the assignment. Don't assume that you can do it all in one go. Break it up into parts and make sure you write tests for anything before you implement it.

Set yourself goals, like having a certain set of functions implemented and tested by a particular date.

Make a plan about how you will implement the ADT, use divide and conquer strategy.

Make sure to properly review your code to improve style and fix any bugs you might find.

Understanding the problem

Read through **the Game Rules** to get a better understanding of how the game works and how you might represent it with code.

Then, read carefully through the **.h** files and try to figure out what information you will need to store to represent a *current state* of the game and how you might want to store it in your ADT.

Hints: How can you represent say a "deck of cards", using a list of cards? You can use `Card.h` and `Card.c` (provided, see below) to represent a card. How about "discard pile", "draw pile", four "hands", etc. What other information do you need to keep in order to follow all the rules for this game? Every "*action*" (move) needs to change a current state of the game based on the "rules". Most of the required information are also derived from a current state of the game. So, it is important to first understand what we need to store and how to represent a state of the game. If you do this properly, your other tasks will be simplified. You may want to define a `struc` to represent a current state, with the required fields to represent all the values we need to represent a state of the game.

Get the files

There are three header files you'll need for this assignment:

- **Card.h** and **Card.c**
- **Game.h**
- **player.h**

You MUST NOT modify these header files! If required, you should declare any additional constants, structures, and prototypes in the .c file that implements them. Make sure you mark your own helper functions **`static`**.

The files that you need to create are:

- **testGame.c**, which tests an implementation of the functions in **Game.h**,

- **Game.c**, which implements the functions described in **Game.h**, and
- **player.c**, which implements a player using the function described in **player.h**.

You can download **Card.c** file listed above and use it, please note that you do not need to submit Card.c file.

The Task

Writing the Game ADT

Your game implementation should be implemented in **Game.c** and implement the functions from **Game.h**. Don't try and implement the ADT in one go, but try and break it up into parts, making sure you have tests for each part before trying to implement it.

Once you have written some tests (see below), you can start testing your implementation. It is important that you continue to write and improve your tests as you work on your Game implementation. This will help you ensure that when you find bugs in your code they get removed for good and as your code gets more complex you can still be sure that it is correct.

Testing the ADT

Start by writing a simple set of tests first. The tests should be implemented in **testGame.c** and test the implementation of the functions in **Game.h**. This will help you get an understanding of how the game works. You should continually work on improving the tests you write throughout the assignment period.

It is incorrect to write tests using inputs that are invalid for a function.

The Player

you need to implement a Player (say using *artificial intelligence*!) to play the game for you. You must implement the function found in **player.h** and your Player can use any of the functions from **Game.h**, except those which specify that they should not be used by a Player, to determine what move it should make.

Your player should not take more than two seconds to decide its move. If your player takes too long to decide on its action, the game will end and your player will be *at fault*.

Testing `Player.c`

~~*[01:10pm 20/Oct]* **Note:** `Game.o` has been updated, please re-copy the latest Game.o to your runner directory:~~

~~*[02:00pm 21/Oct]* **Note:** `Game.o` has been updated again, please re-copy the latest Game.o to your runner directory:~~

[04:30pm 23/Oct] **Note:** `Game.o` has been updated again, please re-copy the latest Game.o to your runner directory:

```
% cp  /home/cs1511/public_html/18s2/assigns/ass2/runner/Game.o     .
```

You can test your `player.c` against the three players (provided) by following the steps below.

First, you may want to create a separate directory, say called `runner`, for testing `player.c`.
Create the directory named `runner` and change to that directory:

```
% mkdir runner
% cd runner
```

Now you can copy the reuired files to your working directory (`runner`):

```
% cp  /home/cs1511/public_html/18s2/assigns/ass2/runner/*.o     .
% cp  /home/cs1511/public_html/18s2/assigns/ass2/runner/*.h     .
```

The above command will copy the following files: `Card.h, Card.o, Game.h, Game.o, GameRunner.o, player.h, player0.o, player1.o, player2.o`.

Now, **copy your `player.c` to the directory `runner`** (the directory you created above).

You now have all the required files in the directory `runner`. From this directory you can create a `GameRunner` by **adding your `player.c`** using the following command:

```
% dcc   -o   GameRunner   player.c   GameRunner.o   Game.o   Card.o  player0.o  player1.o  player2.o
```

You can run `GameRunner` as shown below:

```
%  ./GameRunner
```

If you want to pause moves, provide command-line argument "`--wait`" as shown below:

```
%  ./GameRunner  --wait
```

If your player can successfully play a game and perform reasonably well (you don't need to win!), you will be awarded full marks for automarking of this part.

Just to re-iterate, a game can end in one of the following ways:

- A player plays their final card, winning the game,
- A player is unable to make a move and there is no way to draw a card (such as both the draw and discard piles being empty),
- A player takes too long to make a move,
- A player tries to make an illegal action, or
- A player tries to use a function that a player should not access.

If a player causes to end the game in any way other than by winning the game or a player being unable to draw a card, they will be considered *at fault*.

Submitting

You need to submit the following three files:

- *Game.c*
- *testGame.c*
- *player.c*

The submission system will compile all three files, and test your `Game.c` with `stage2.c` (provided in the specs).

If any of the above is not successful, it will be reported. Carefully read the output log, you should address the problems and later resubmit all three files. Please note that you can submit multiple times, **every time you need to submit all three files**. We will mark your latest submission.

If you prefer, you can submit files with "errors", however, you are unlikely to receive any marks for such submissions.

Go to : Assignment-2 Submission

Late Penalty

The assignment is due at **23:59** on **Friday of Week-13**, Sydney time.

You will not be able to submit after 23:59 Saturday 27/Oct, 10% late penalty will apply per day for a late submission.

It is essential that you consistently work on the assignment, and at the end of each week (as mentioned above) submit the progress you have thus far.

This is not an assignment that you can just rush through immediately before the due date: you will need to demonstrate consistent, incremental improvement throughout the period of the assignment.

Important: You need to submit your working progress by submitting all three files at the end of week-10, week-11, week-12 and of course final submission before the due date.

A penalty of 3% of the marks awarded to you for this assignment will be applied for each missed submission (for weeks 10, 11 and 12).

Assessment

This assignment will contribute 13% to your final mark.

Mark breakdown

Your combined mark for correctness + performance is capped at **80%**.

50%	**Game.c** correctness
20%	**testGame.c** correctness
15%	**player.c** performance
20%	Code style

You may have noticed that the percentages above add to more than 100%. This is because your combined mark for correctness+performance (testGame.c, Game.c, player.c) is capped at 80%, which means that you can still get full marks for correctness without needing your player to be at the top of the leader-board.

Plagiarism

Keep in mind that if you “cheat” on this assignment, the only person you’re cheating is yourself. We give you these assignments so that you can gain experience and develop new skills. If you plagiarise, you’re throwing away a great opportunity, and letting down your team-mates.

We will adhere to the very strict Plagiarism policy of UNSW.