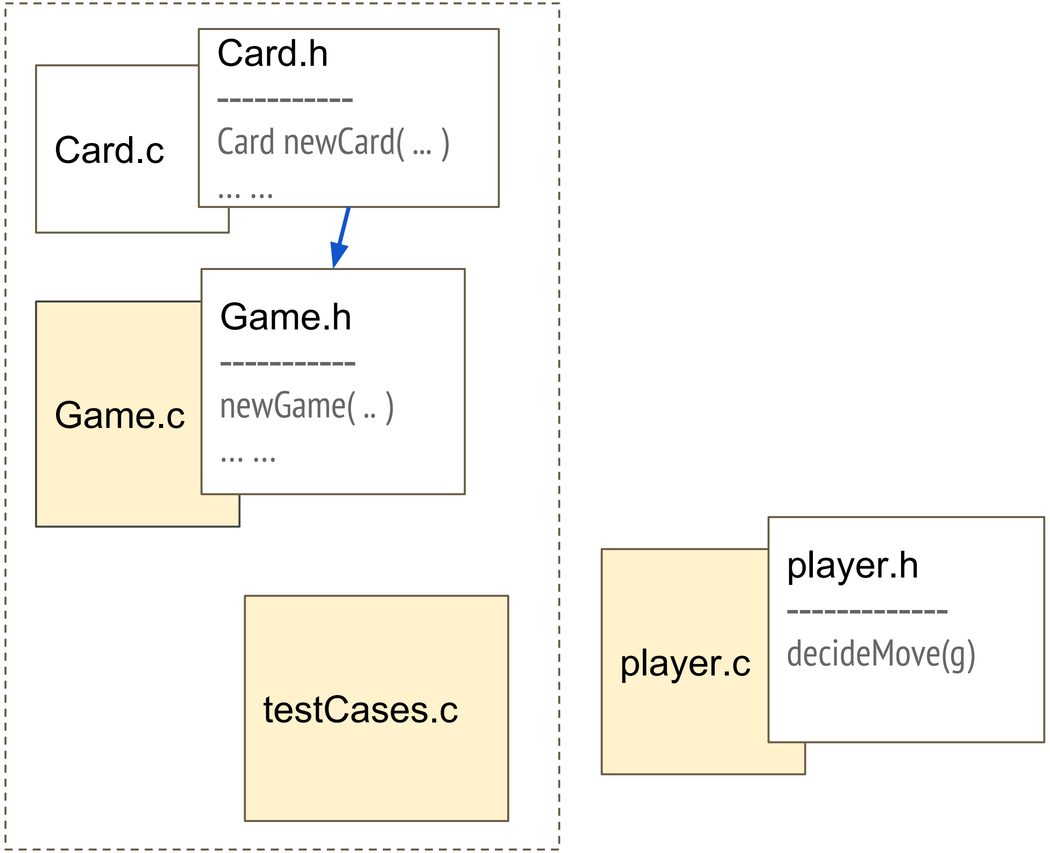
COMP1511:
Assignment 2 - Getting Started

Session 2, 2018

Assignment-2



(you do not need to implement this)

GameRunner.c

```
Game    g = newGame( ... )

// for player 0
playerMove    move = decideMove(g)
while(move is not END_TURN){
        playerMove   move = decideMove(g)
        ...  ...
}

// for player 1
playerMove    move = decideMove(g)
while(move is not END_TURN){
        playerMove    move = decideMove(g)
        ...  ...
}
... ...
```

Assignment-2 : Game.c

- Need to **store information** regarding the current state of a game (for example, linked list for stake, queue, etc.). In Ass2, you need to store information on many entities, like :
    - deck (linked list),
    - discard pile (linked list),
    - 4 hands of 4 players (linked list),
    - **and others** - now you need to think what you need!

- Need to **implement "actions" / "move"** that change the current state. For example, draw a card, discard a card, play card with value 2, play card with other special values, etc. These actions should accordingly change the game state.
- Need to **implement** all the required **functions** defined in **Game.h** ADT.

Assignment-2: testGame.c

- Start by writing a simple set of tests in **testGame.c**
- The tests should test the implementation of the functions in Game.h.
- This will help you get an understanding of how the game works.
- For example,
    - stage-1: create a new game and distribute cards to 4 players. Now test the deck and cards in 4 hands, they should match the expected cards.
    - stage-2: make few moves/actions, and test the outcomes using functions available in Game.h like getDeckCard, getDiscardPileCard, getHandCard, etc.
    - **Add more stages** … ...
- You should **continually** work on **improving** the tests you write throughout the assignment period.

Assignment 2: Testing the Game ADT (`testGame.c`)

The following webpage offers useful tips on “how to get started” for developing test cases for your `testGame.c`

Goto: https://cgi.cse.unsw.edu.au/~cs1511/18s2/assigns/ass2/intro_to_testing/index.html

Helper / Additional Functions

- You can add “helper” (additional) functions in your files.
  Make sure to **declare additional functions as “static”**
- In fact, you should identify “repetitive” / “similar” tasks, factor out common components and write functions that you could use as helper functions.
- This will also increase readability of your code.
- If your function is **too long**, stop and think!
  Can you divide it into sub tasks and write functions for sub tasks.

Enumeration Type

- An ***Enumeration type*** allows a programmer to define and name a ***finite set of named constants*** (called **enumerators**).
- Enumeration types are mainly used to **improve** program **readability**.
- For example, we can define enumeration type for “Days” as below:

```
enum  Days { sun, mon, tue, wed, thu, fri, sat };
 …  …  …
enum  Days d1, d2 ;  // declares variables d1 and d2
```

Enumeration Type

- Alternatively we can also use **`typedef`** to define “Days” as below:

```
typedef enum  { sun, mon, tue, wed, thu, fri, sat } Days;

…  …  …

Days d1, d2 ;  // declares variables d1 and d2
```

- d1 and d2 can **only be assigned values** from the set of enumerators defined as **`Days,`** for example:

  d1 = wed ;   d2 = mon;

- More examples, from **`Card.h`** :

```
// The various colors that a card can have.
typedef enum {RED, BLUE, GREEN, YELLOW, PURPLE } color;

// The various suits that a card can have.
typedef enum {HEARTS, DIAMONDS, CLUBS, SPADES, QUESTIONS} suit;
```

Enumerators

- Enumerators (sun, mon, etc.) are **constants** of type **int**.
- By default, the first one is given the value 0, and each succeeding one has the next integer.

```
typedef enum  { sun, mon, tue, wed, thu, fri, sat } Days;
                0    1    2    3    4    5    6
```

- We can also initialse enumerators, for example

```
typedef enum  { sun=1, mon, tue, wed=7, thu, fri, sat } Days;
                1      2    3    7      8    9    10
```

- Named constants need to be unique, int values need **not** be unique.

Testing: Four types of Testing

- **Bad Testing !**
  - “I’ve written this program… now, let’s write some tests… my program passed, woohoo!”

- **Black box tests**
  - “I’ve written this program, now let’s get someone else to test it for me!”
  - your program is a magical **black box**, where information goes in, and information comes out.
- **White box tests**
  - “I’ve written this program, can you **look through it**, and check it’s right?”
- **Unit tests**
  - “As well as testing my whole program, I’ll test **each of the small parts** of it.”
  - faster and easier to check our small units and then check the whole program

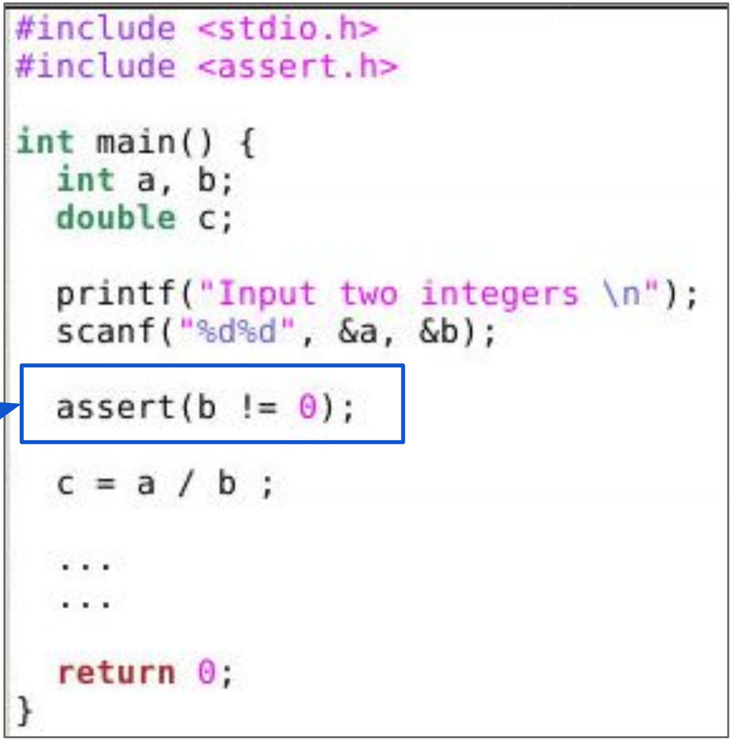
Testing with `assert`

- `assert` is a macro, to use it you must **include** the header file "**assert.h**"

- `assert` is used to check specific conditions at runtime, useful for testing and debugging a program.

-

- Often an expression is a **boolean** condition.



```
#include <stdio.h>
#include <assert.h>

int main() {
  int a, b;
  double c;

  printf("Input two integers \n");
  scanf("%d%d", &a, &b);

  assert(b != 0);

  c = a / b ;

  ...
  ...

  return 0;
}
```