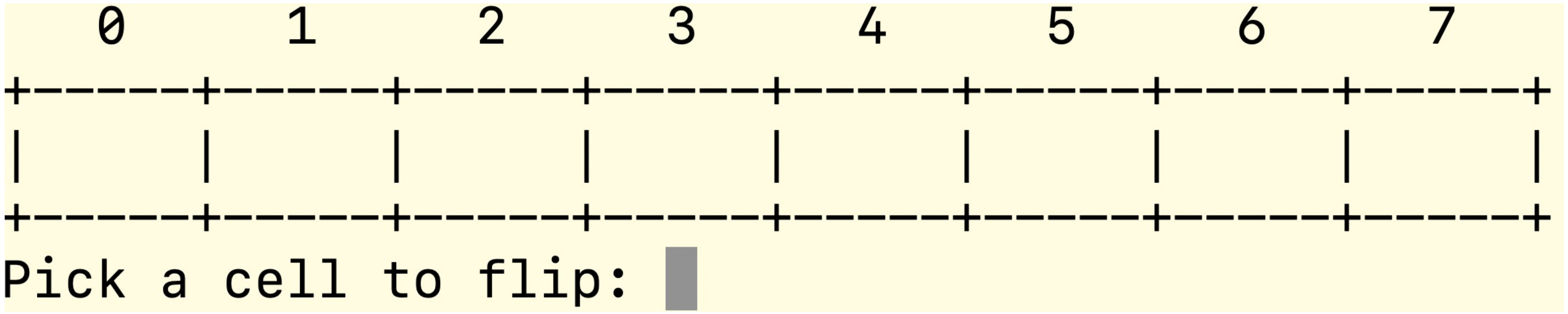
Project: Memory Game

Outline

- Introduction of rules
- Design
- Code skeleton

Introduction

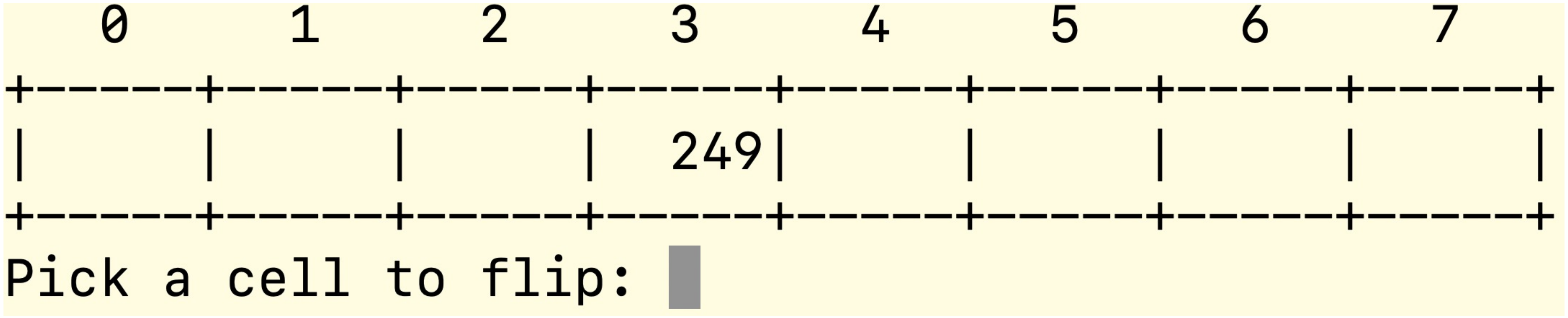
- Two-face cards with content-side facing down.
  - Here is an example of 8 cards.



```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 
```

The first flip in a round

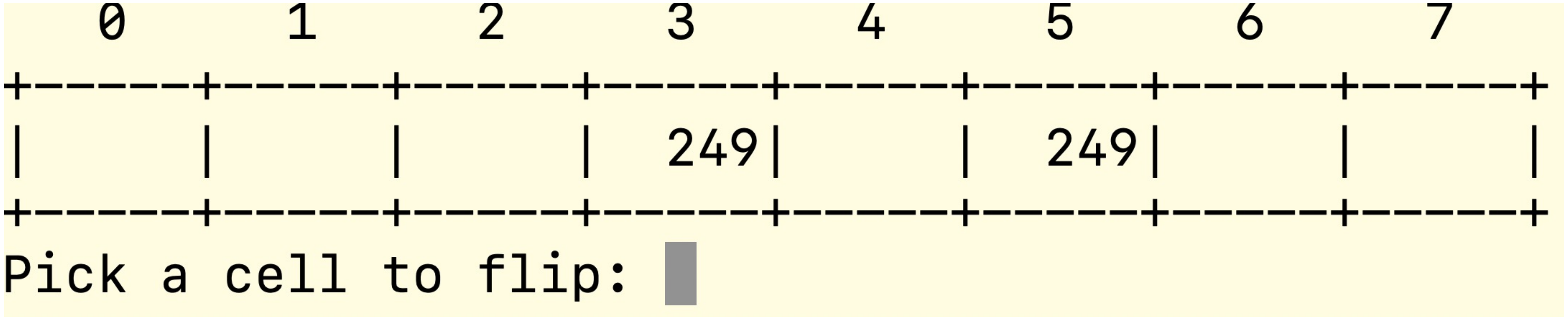
- In each round, flip two cards.
  - Flip the card by entering an integer representing an index, starting from 0. In our example, the last index is 7, so the input should be in [0, 7]. Suppose we select 3.
  - After the first flip, the content of the card, which can be an integer, is shown.



```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |  249|     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 
```

Second flip in a round match to first flip

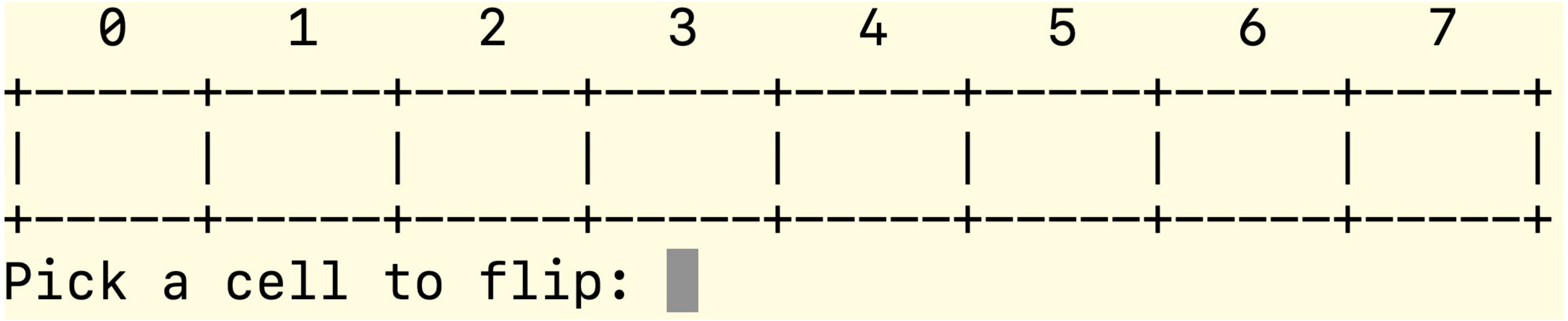
- Afterwards, flip the second card.
  - If the integer of the second card matches that of the first one, the second card is opened; otherwise, close both cards.
  - Suppose we choose 5, since both space 3 and 5 have the same number, we find a match.



```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |  249|     |  249|     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 
```

Second flip in a round not to match to first flip

- Suppose we choose card 6 after we flip card 3 for the first time. Since Card 6 does not have the same number as that of the first flip, ie, Card 3, in this round, both cards are turned.



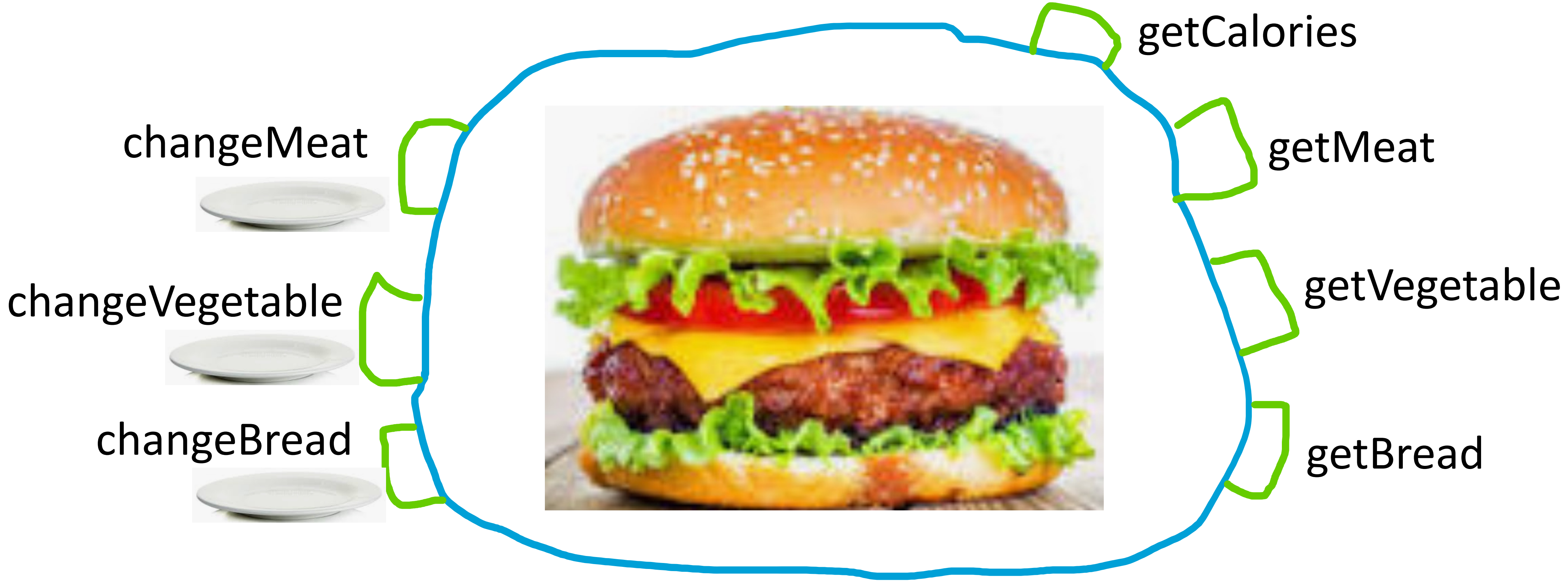
```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 
```

Continue until all matched numbers are found

- To make the game challenging, add cards whose sides are both empty.
  - In a round, if the first flip shows an empty face (ie, no integer disclosed), then this card has two empty faces. In this case, we cannot tell whether the second card flipped has an integer or not.
- Stop until all matched twin integers are found.
- Report the total number of flips to find out all matches.
  - Print out “Congratulations! Take … steps to find all matched pairs.”, where … is the number of steps (ie, flips).

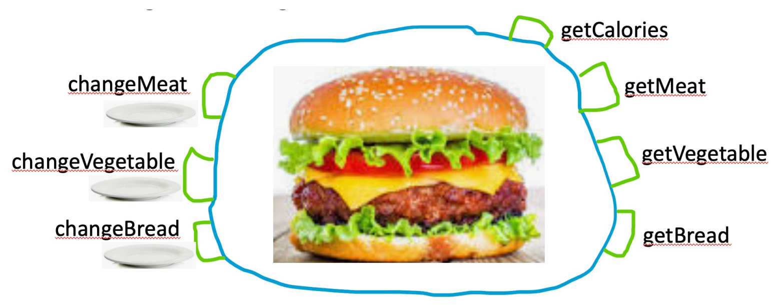
Object Oriented Programming

an intelligent hamburger



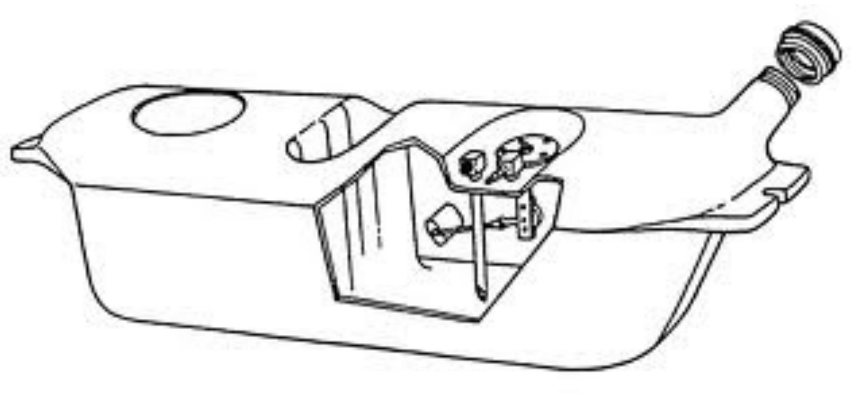
Data and operations for an intelligent hamburger

- Data of a hamburger: bread-, vegetable- and meat- layer.
- An intelligent hamburger combines data with operations on those data.
  - Operations to access data
    - getBread, getVegetable, getMeat, getCalories
  - Operations to change the data
    - changeBread, changeVegetable, changeMeat
    - Why is there a plate besides those changeXYZ buttons? For example, in changeBread button, need to use a plate (parameter) to hold the new bread layer, which is used to replace the current bread layer.



Class and Object

- A class is the blueprint from which objects are made.
  - For example, class as blueprint of fuel tank of a car, while objects as the implemented fuel tanks.



Data member vs. methods

- Things object knows are data members
- Things object does are methods
  - Object
    - fuel tank
  - Data members
    - Current gas level
  - Methods
    - Fill tank with more gas
    - Send gas through the gas lines
    - Get current gas level of tank

Encapsulation

- Combine data members and methods in one package

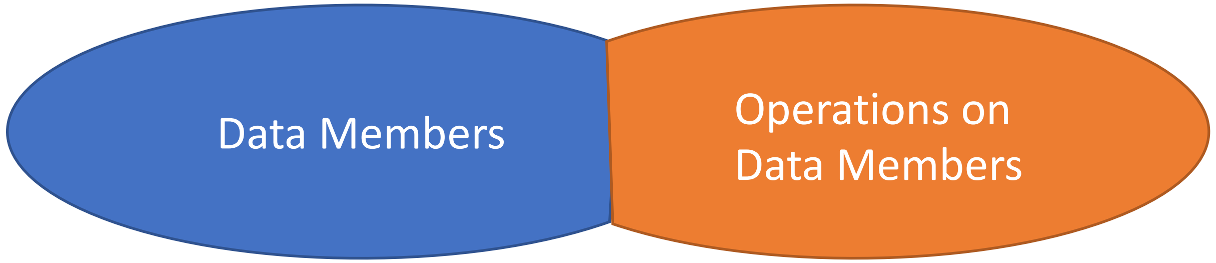
current gas level
Fill tank with gas
Send gas through gas line
Get current level of tank

- Encapsulation is also called information hiding
  - encapsulation hides the implementation details from the user of the object.
  - One can use a fuel tank without knowing how it works.

Do you need to be an auto engineer to operate the car?

Take home message – what is class?

- class is the encapsulation of data members and methods performed on those data members.
- Encapsulation: class = data member + operations on those data members



- Next task:
    - Define class MemoryGame
    - Construct an object game from class MemoryGame
    - Use object game

Data members of Memory Game

```
private: //private data members, private means that
         //only methods in this class, not other class,
         //can access or modify these data members.
    int numPairs; //numPairs of identical twin items
    int numSlots; //size of array value, besides identical twins,
         //may contain empty string to make the problem more challenging
    string *values; //a string to represent the layout of data,
        //mixed with possible empty strings.
        //Use array to access each element in const time.
```

Operations for data in Memory Game

```
public:
    MemoryGame(); //default constructor, with 3 pairs of numbers
        //randomly located in 8 blocks (two blocks are empty).
    ~MemoryGame(); //destructor
    void play(); //play the game
    void display(bool* bShown); //display array values.
        //if bShown[i], where i is index, is true,
        //them values[i] is displayed, otherwise, values[i] is not displayed.
```

Tasks of the Project

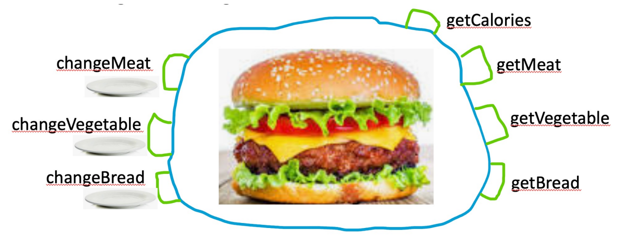
- Define functions swap and randomize, which will be used by class MemoryGame but they are not method members of MemoryGame.
- Implement MemoryGame, finish its methods
    - Default constructor MemoryGame(): initialize data members
    - Destructor ~MemoryGame(): release dynamic memory
    - Method play: simulate the playing of game
    - Method display: display cards
- Test MemoryGame class.

Constructor of a class

- Constructor of a class is to initialize the data members. It create an object with data member initialized, and attach method members for this object.
- Constructor(s) have exact the same name as class, case to case, letter to letter.
- Constructor(s) have no return type, not even void.
- Default constructor has no parameter.

Constructors of Intelligent Hamburger

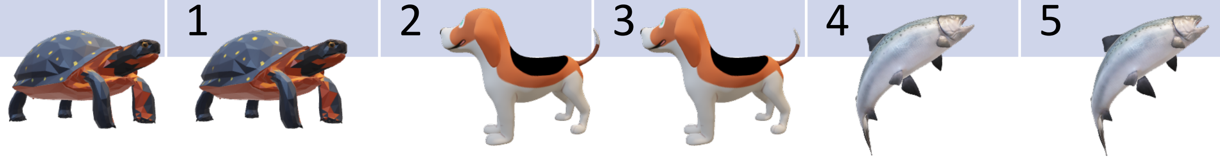
- The default constructor is a cashier who makes a ”typical” hamburger without taking ”individualized” request from users of the class.
    - For example, a ”typical” hamburger has wheat bread, beef, lettuce and onion.
- A non-default constructor takes parameters to “individualize” an hamburger. Say, one might like chicken instead of beef.
- A constructor (cashier) creates a hamburger with those ”typical” layers, add operations (method members) to make it intelligent.
    - An intelligent hamburger has data (bread layer, meat layer, and vegetable layer) and operations (getBreadLayer, changeBreadLayer, getCalories, …). It like to add buttons as operations.



Randomize an array

- Purpose: get a permutation of indices, each one appear once and exactly once. Then randomize it.
- First, get a permutation of indices. Suppose there are 8 of them.

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7



- Put the first pair of integers in indices 0 and 1, then the second pair of integers in indices 2 and 3, and the last pair of integers in indices 4 and 5. The last two cells put nothing.
  - Such layout is not challenge at all. But, wait until we permutate the array.

Randomize an array: II

- Pick a random int in [0, 7], which are indices. Suppose we pick up 5.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	1	2	3	4	5	6	7

- Swap the elements at 5 and at the last index (so that an element will not get pick up twice).

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	1	2	3	4	7	6	5

randomize elements in this segment

Randomize an array: III

- Pick up a random int in [0, 6], with the first pick up is put in index 7.

0	1	2	3	4	5	6	7
0	1	2	3	4	7	6	5

- Suppose we pick up 5 again, swap the elements indexed at 5 and 6.

0	1	2	3	4	5	6	7
0	1	2	3	4	6	7	5

randomize elements in this segment

- Continue until we finish randomize the segment with the first two indices.

Use randomized array for layout

- Suppose somehow we get a randomized array as follows (not related with the steps of randomization in the previous two slides).

0	1	2	3	4	5	6	7
4	1	7	5	6	2	0	3

- Remember the original layout without randomization.

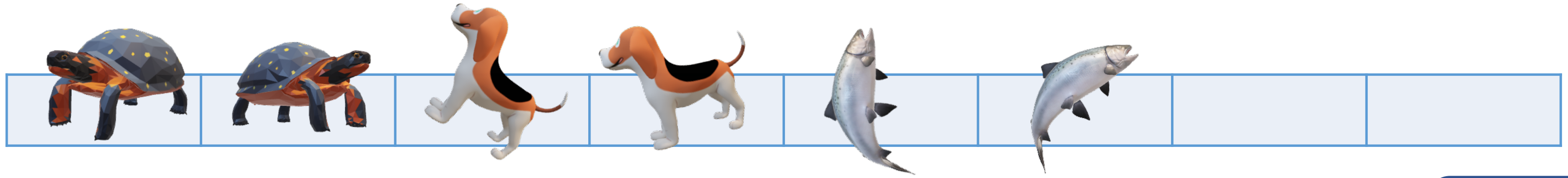
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

Put elements with randomized layout

- Purpose: put elements using **randomized** permutation of indices.

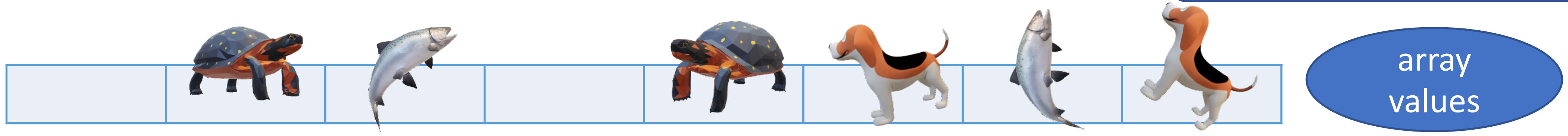
0	1	2	3	4	5	6	7
4	1	7	5	6	2	0	3

- Layout elements (ie, animals) without using randomized array.



- Layout elements using the above randomized array.

Change animals a little to show which animal is put in which index

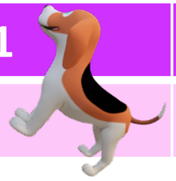


Put elements with randomized layout: II

- a randomized array

0	1	2	3	4	5	6	7
4	1	7	5	6	2	0	3

- Elements to be randomized, saved in an array or vector.

0	1	2
		

- array to hold randomized elements (including empty string).

0	1	2	3	4	5	6	7

array values

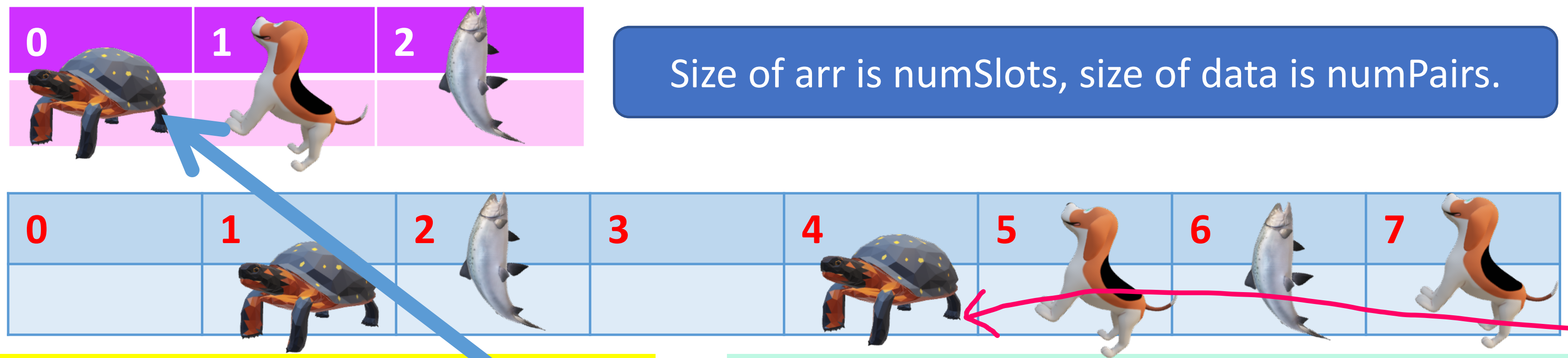
Put elements with randomized layout: III

arr

0	1	2	3	4	5	6	7
4	1	7	5	6	2	0	3

data

values



Size of arr is numSlots, size of data is numPairs.

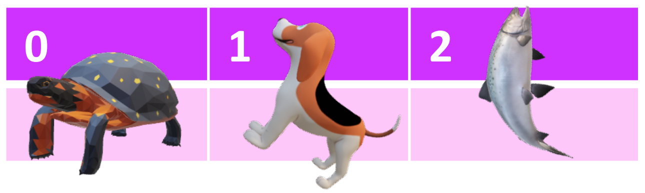
```
values[arr[0]] = to_string(data[0]);    values[arr[1]] = to_string(data[0]);
values[arr[2]] = to_string(data[1]);    values[arr[3]] = to_string(data[1]);
values[arr[4]] = to_string(data[2]);    values[arr[5]] = to_string(data[2]);
values[arr[6]] = “”;                    values[arr[7]] = “”;
```

Put elements with randomized layout: IV

arr

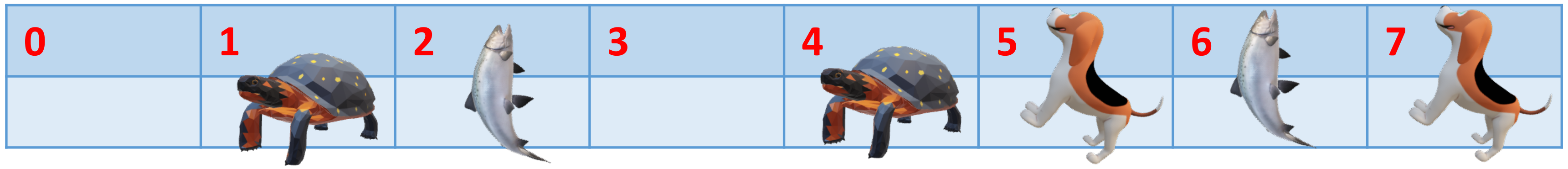
0	1	2	3	4	5	6	7
4	1	7	5	6	2	0	3

data



Size of arr is numSlots, size of data is numPairs.

values



```
values[arr[0]] = to_string(data[0/2]);          values[arr[1]] = to_string(data[1/2]);
values[arr[2]] = to_string(data[2/2]);          values[arr[3]] = to_string(data[3/2]);
values[arr[4]] = to_string(data[4/2]);          values[arr[5]] = to_string(data[5/2]);
values[arr[6]] = “”;                            values[arr[7]] = “”;
```

Put elements with randomized layout: V

arr

0	1	2	3	4	5	6	7
4	1	7	5	6	2	0	3

data

0	1	2

Size of arr is numSlots, size of data is numPairs.

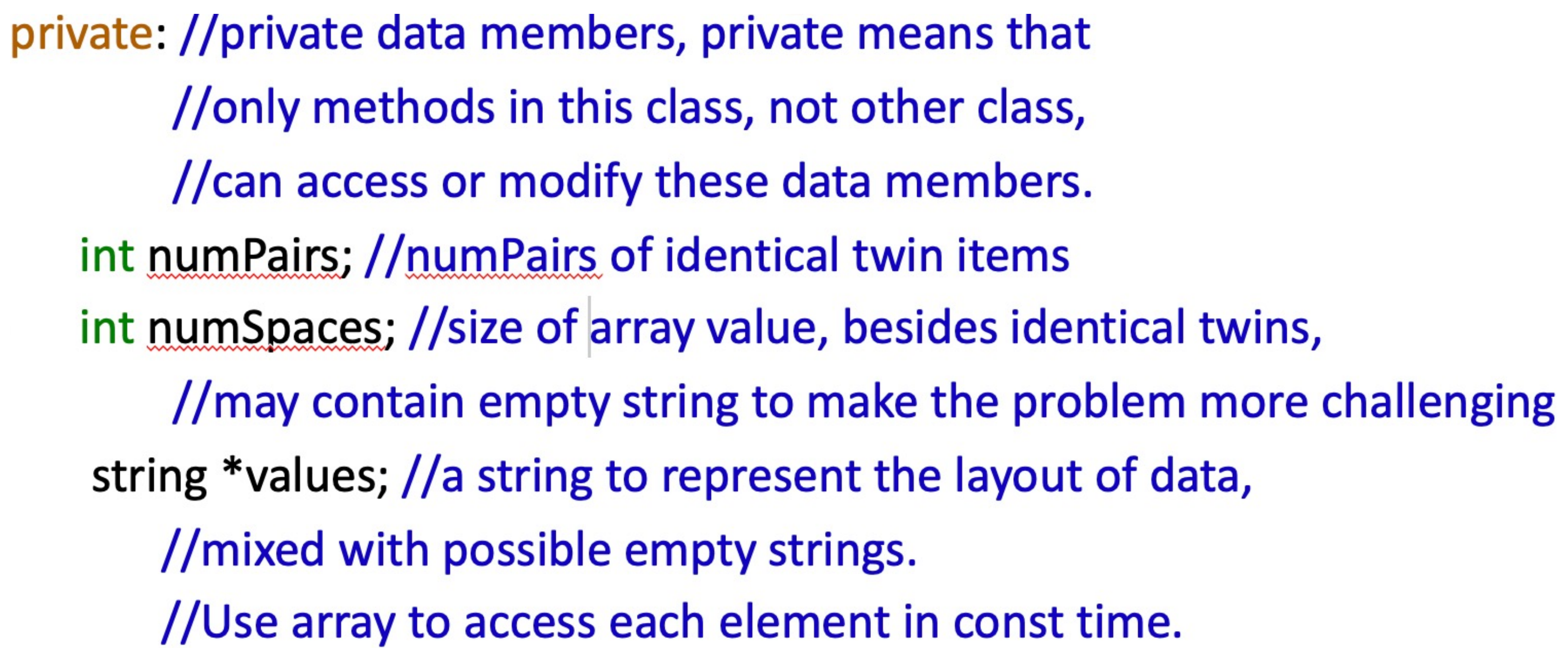
values

0	1	2	3	4	5	6	7

```
for (int i = 0; i < ?; i++)
    if (i < ??)
        values[arr[i]] = to_string(data[???]);
    else values[arr[i]] = “”;
```

Default Constructor of MemoryGame

- Set numPairs to be 3.
- Set numPlaces to be 8, holding numPairs * 2 ints and 2 empty strings.
- Set values to set ints and empty strings randomly (steps listed in previous slides).



```
private: //private data members, private means that
         //only methods in this class, not other class,
         //can access or modify these data members.
  int numPairs; //numPairs of identical twin items
  int numSpaces; //size of array value, besides identical twins,
         //may contain empty string to make the problem more challenging
  string *values; //a string to represent the layout of data,
        //mixed with possible empty strings.
        //Use array to access each element in const time.
```

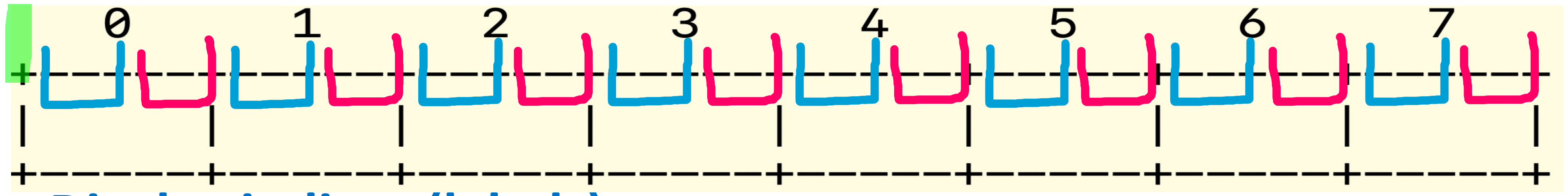
Method display of MemoryGame

- Use an array parameter to decide whether an item is displayed or not.

```
   0      1      2      3      4      5      6      7
+------+------+------+------+------+------+------+------+
|      |      |      |      |      |      |      |      |
+------+------+------+------+------+------+------+------+
```

- **Display indices (labels).**
- Display separate line.
- **Display data if the corresponding bShown value is true, or display “”.**
- Followed by another separate line.

Method display of MemoryGame: II



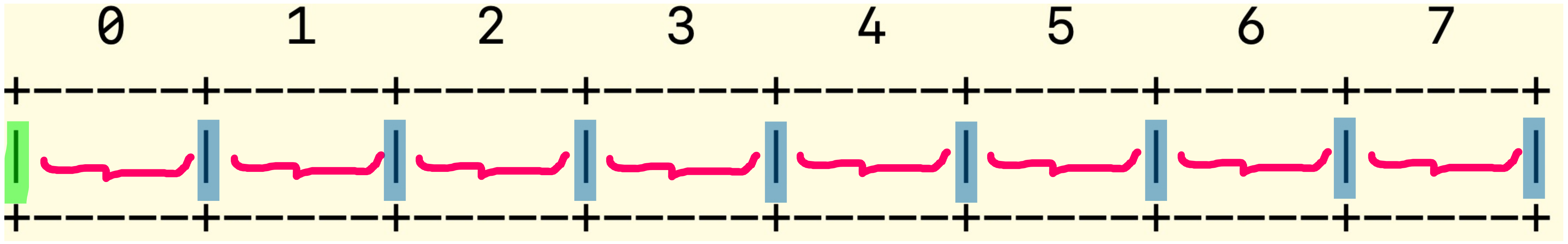
```
    0      1      2      3      4      5      6      7
+------+------+------+------+------+------+------+------+
|      |      |      |      |      |      |      |      |
+------+------+------+------+------+------+------+------+
```

- **Display indices (labels).**

▮ display 1 space, ⊔ display index in 3 spaces, ⊔ display 3 spaces

```
//display label
cout << " "; //one space before the first label
for (int i = 0; i < numSpaces; i++)
{
     cout << setw(3) << i;
     cout << setw(3) << " ";
}
cout << endl;
```

Method display of MemoryGame: III



```
    0      1      2      3      4      5      6      7
+------+------+------+------+------+------+------+------+
|      |      |      |      |      |      |      |      |
+------+------+------+------+------+------+------+------+
```

- **Display data if the corresponding bShow value is true, or display “”.**

Display values[i] based by bShown[i] is true or false, where 0 <= i < numSlots.

```
if (bShown[i])
   cout << setw(5) << values[i];
else cout << setw(5) << "";
```

Test display method in main function

```
bool bShown[8];

for (int i = 0; i < numSlots; i++)
    bShown[i] = true;

MemoryGame game; //#include "MemoryGame.hpp" before int main()
game.display(bShown);

return 0;
```

How to test your code

- Put the follow files, saved in blackboard, in one directory.
  - MemoryGame.hpp (no modification is needed),
  - MemoryGame.cpp (finish TODO part),
  - MemoryClient.cpp (no modification is needed),
  - makefile
- Under terminal, type in make and return key.
- Run ./memory with return key.
- You should see something like this (result may vary).

```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|  807|  807|     |  249|   73|  249|     |   73|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Define play method

- bool array **bShown**, with size numSlots, indicates which cell is displayed and which is not.
- Suppose the contents of **bShown** is as follows.

0	1	2	3	4	5	6	7
true	true	false	true	false	false	false	false

- The corresponding layout of game is as follows.

```
    0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|  807|  807|     |  249|     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Define play method: II

- Key is to manipulate ***bShown*** array.
- Before the first round, each element of ***bShown*** is set to be ………..
- Flip cards until all matched pairs are found.
- Besides ***bShown***, use the following variables.
    - ***index***: the index of the current card being flipped
    - ***numFlips***: number of flips to find all matched pairs
    - ***pairsFound***: number of matched pairs found so far
    - ***first***: index of the first card flipped in a round. Each round

Outline of play function

```
Set variable pairsFound to be zero.
Set variable numFlips to be zero.
As long as pairsFound < numPairs
begin
        Choose a valid index whose cell is not displayed yet.
        flip that card (what to do in a flip, see next slide)
end
Report numFlips taken to find all matched pairs, ended with a newline.
```

What do we do in each flip?

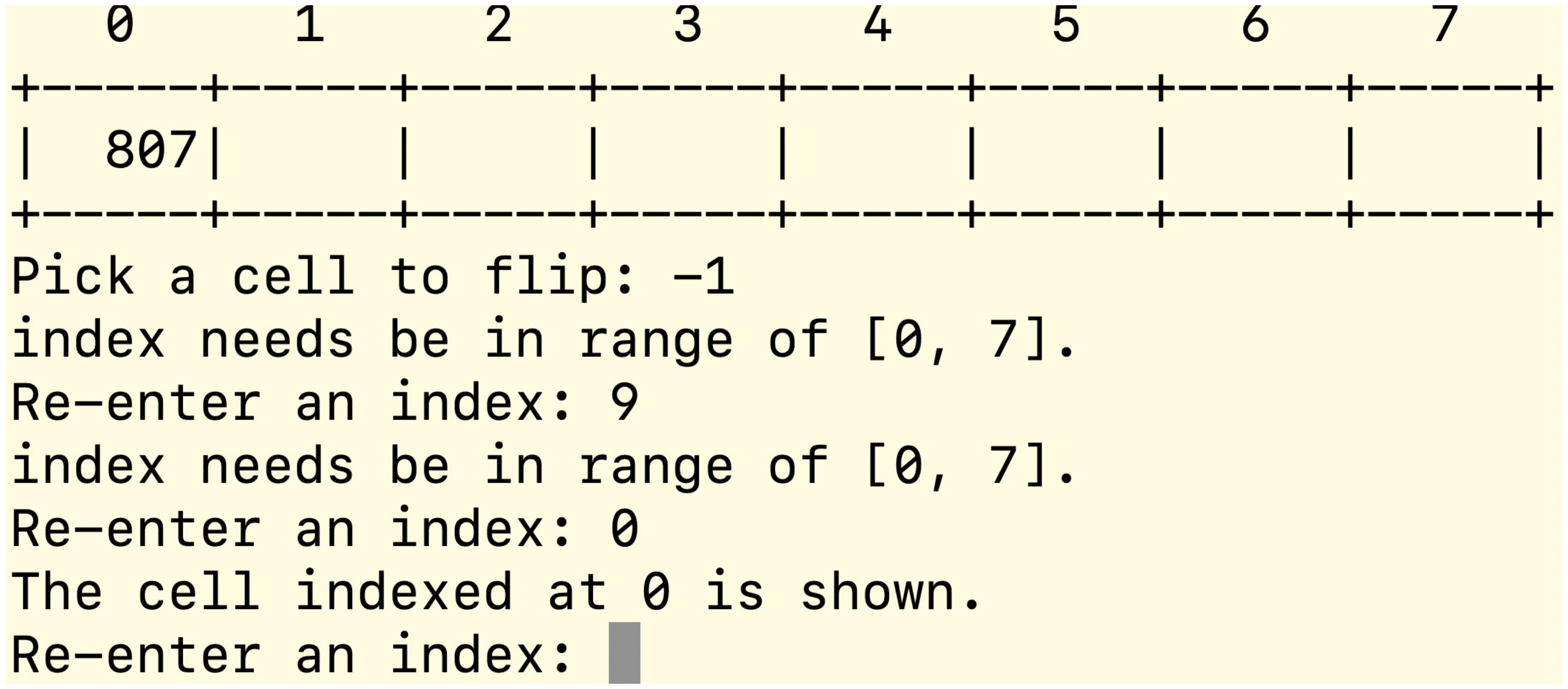
```
if (it is the first flip in a round)
begin
    Set the corresponding *index* of *bShown* to be true.
    Save the chosen *index* to variable *first*.
end
else begin //this is the second flip in a round
        if the second flip matches the first flip and they are not empty string,
           set element of *bShown* to be true, increase *pairsFound* by 1,
        else set element of *bShown* at index *first* to be false.
    end
Display the layout.
Increase *numFlips* by 1
```

How do we know this is the first or the second flip?

More details

When input an index, make sure that it is valid, that is, in range [0, numSlots-1]. Furthermore, the cell is not shown yet.

For example, after flipping cell indexed at 0, enter index for the next flip.



```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|  807|     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: -1
index needs be in range of [0, 7].
Re-enter an index: 9
index needs be in range of [0, 7].
Re-enter an index: 0
The cell indexed at 0 is shown.
Re-enter an index: 
```

Tricks to pass gradescope

- Call int* randomize(int numSlots) AFTER generating three random integers, otherwise, randomize function call rand() first. As a result, the random number generated will be different from those generated before any method, randomize function in this example, calls rand().

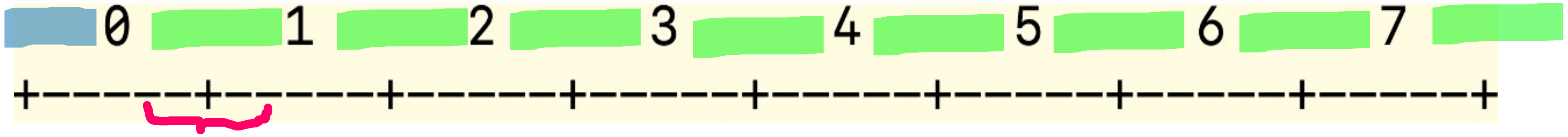
Tricks to pass autograder: use exact wording

- **Pick a cell to flip:**
  - There is one space after : symbol.
- **index needs to be in range [0, ...].**
  - … is the last index of the blocks. The first letter of index is NOT capitalized.
- **The cell indexed at … is shown.**
  - … is the input index.
- **Re-enter an index:**
  - There is one space after : symbol.
  - Use "Re-enter ~~the~~ an index: ". NOT "Re-enter the index: " in the class lecture.
- **Congratulations! Take … steps to find all matched pairs.**
  - … is the number of flips

Tricks to pass autograder: space matters

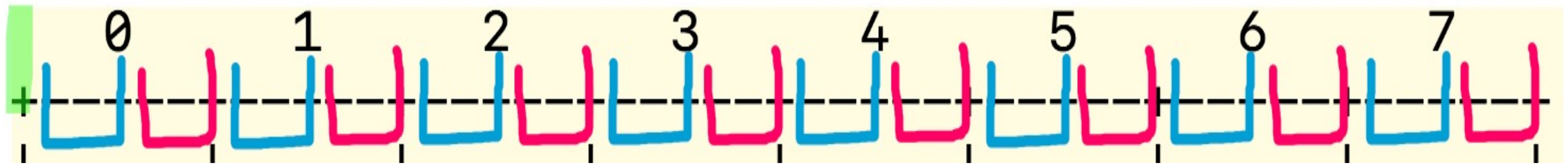
The number of spaces following a number is also fixed.

- For example, students may display the labeling of indices as follows.
- Print three spaces before first index.
- Print an index, followed by 5 spaces, which separated two indices.



```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
```

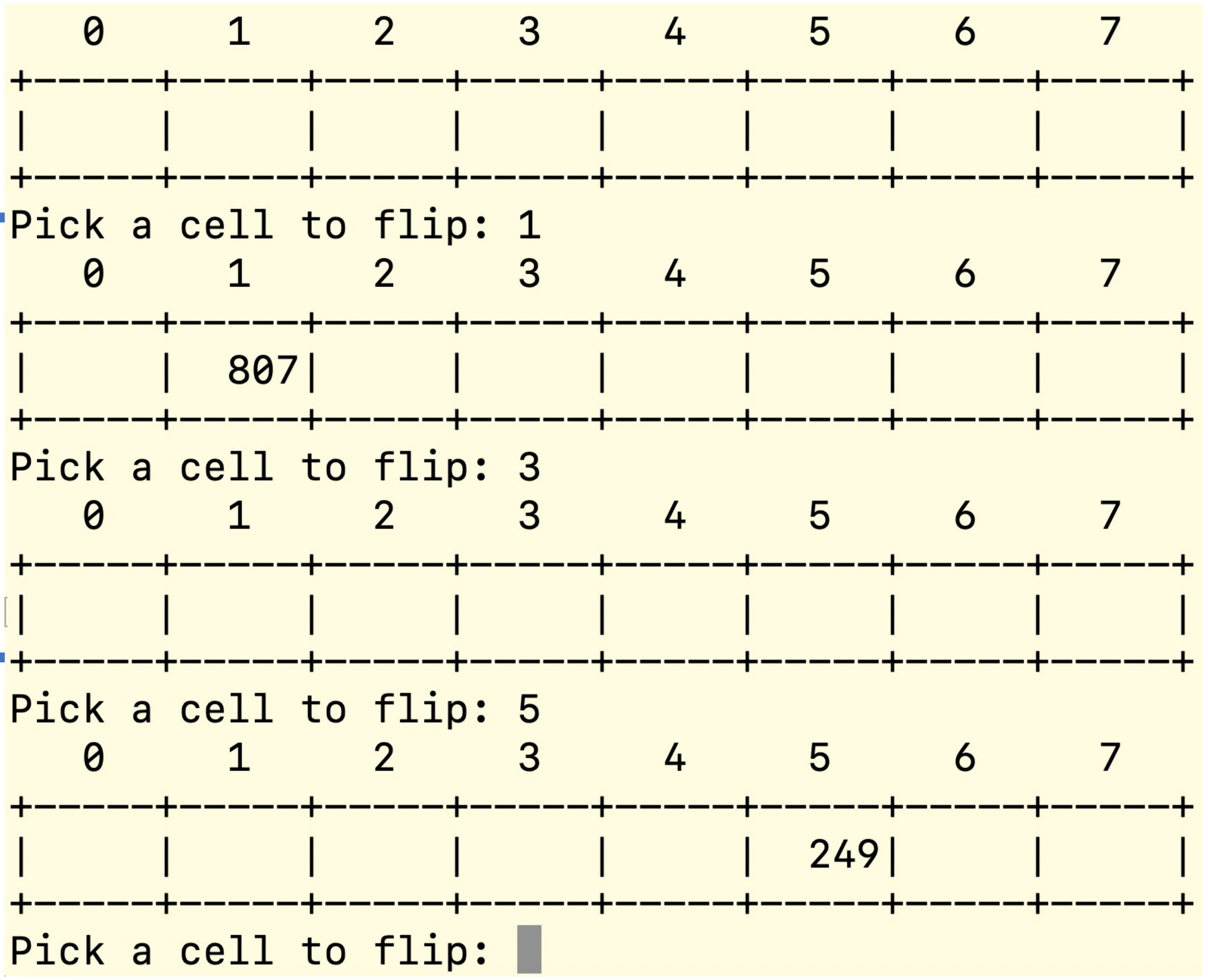
- What is the difference between these two figures?



```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Optional Improvement: clear screen

Output share the same screen, so previous flips results are shown.



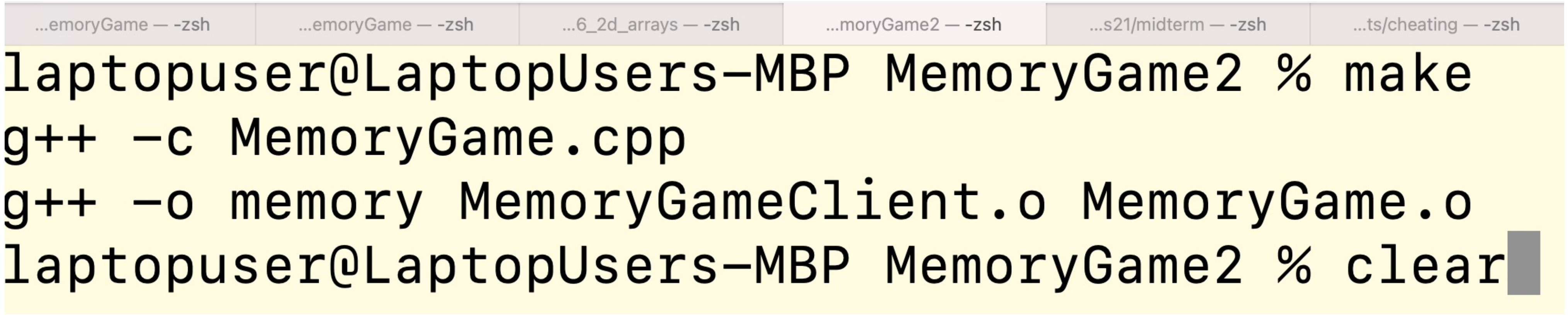
Round 1

```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 1
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |  807|     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 3
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 5
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |  249|     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Pick a cell to flip: 
```

- In first round, first flip cell indexed at 1, turns out to be 807.
- In the second flip of the first round, choose 3, not a match.
- Conclusion: the cell indexed at 3 is not 807.
- In the second round, first flip cell indexed at 5. Because the previous flips in screen, we know to avoid flip card indexed at 1.
- This type layout shows previous flips, which is not challenging enough.

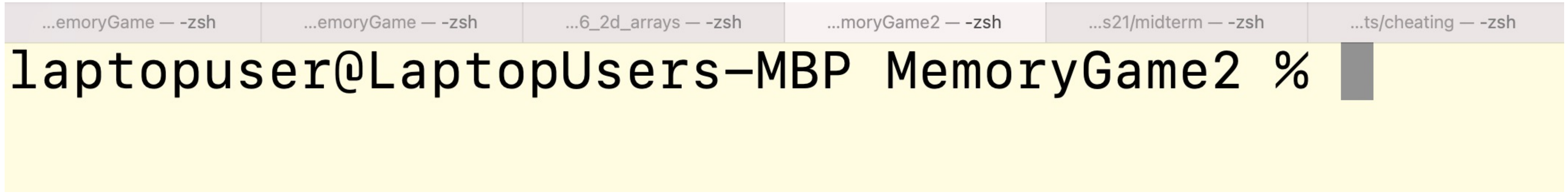
use clear method from Linux to flip screen

- Run commands in Linux, their outputs are shown in the screen.
  - For example, run make command of our project.



```
laptopuser@LaptopUsers-MBP MemoryGame2 % make
g++ -c MemoryGame.cpp
g++ -o memory MemoryGameClient.o MemoryGame.o
laptopuser@LaptopUsers-MBP MemoryGame2 % clear
```

- Now run command **clear**. See what happens?



```
laptopuser@LaptopUsers-MBP MemoryGame2 %
```

Call Linux command in C++

Call setenv method to handle “term not set” error in autograder. Then call Linux command like clear using system method.

```
setenv("TERM", "${TERM:-dumb}", false); //call only once
system("clear");
```

Reference:

https://stackoverflow.com/questions/16242025/term-environment-variable-not-set

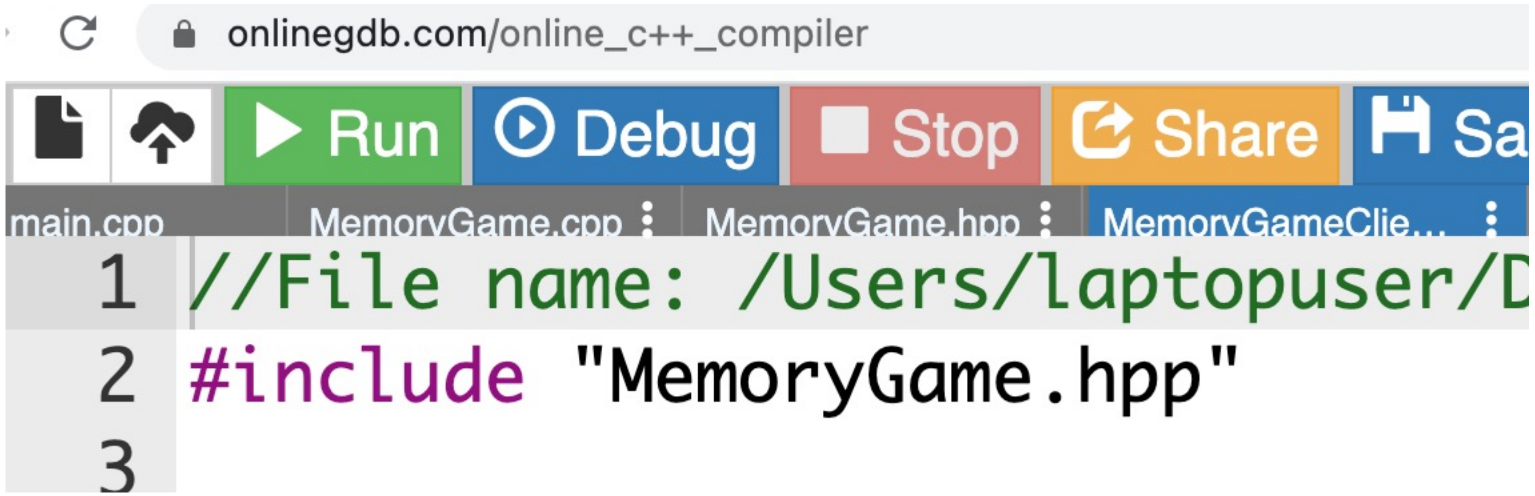
https://stackoverflow.com/questions/19425727/how-to-remove-term-environment-variable-not-set

Run code in onlinegdb C++

- Remove the code in the main method attached.
- Upload MemoryGame.hpp, MemoryGame.cpp, and MemoryGameClient.cpp.
  - MemoryGame.hpp: header file, it is like declaration of a class with its data members and functions.
  - MemoryGame.cpp: source code, implement the functions declared in MemoryGame.hpp.
  - MemoryGame.cpp generates MemoryGame objects. This is like a factory branch to produce products.
  - MemoryGameClient.cpp: test MemoryGame objects. This is like a Quality Analysis branch to test products.

Run projects in onlinegdb C++: II

- Remove the contents of the original main.cpp in onlinegdb project.
- Upload MemoryGame.hpp, MemoryGame.cpp, MemoryGameClient.cpp.
- Click Run button.



```
//File name: /Users/laptopuser/D
#include "MemoryGame.hpp"

```