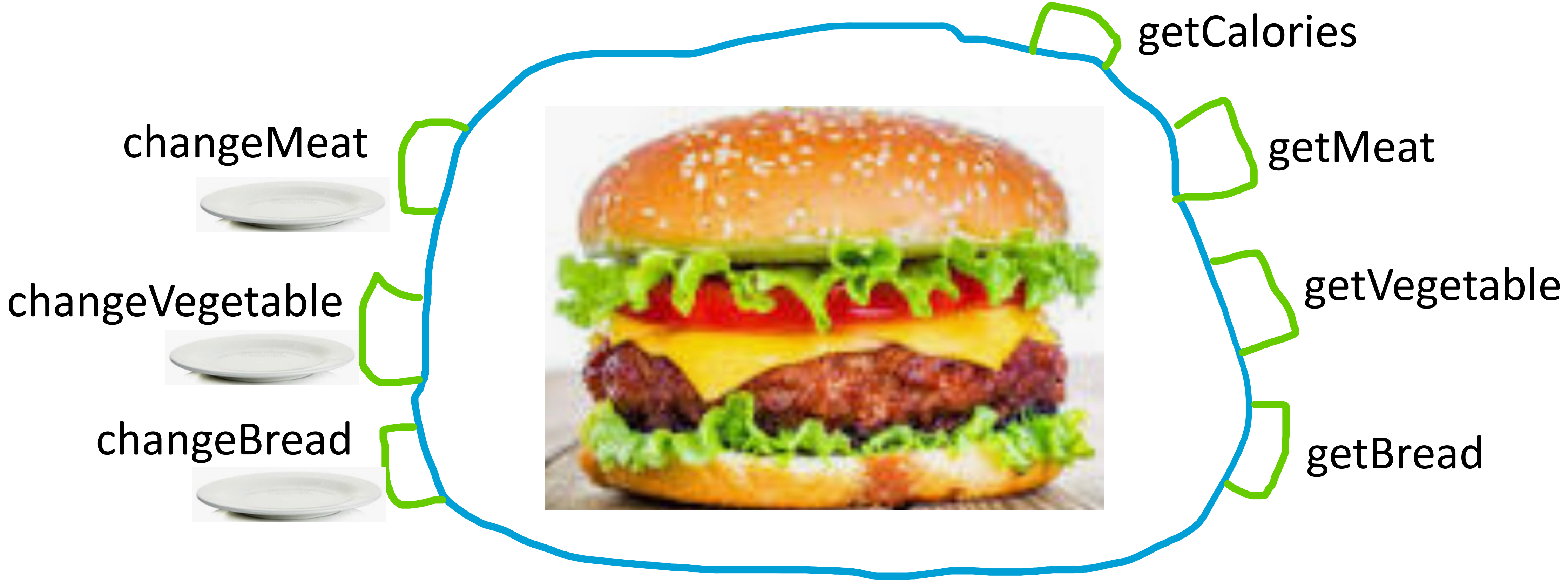
Project: Memory Game

Outline

- Introduction of rules
- Design
- Code skeleton

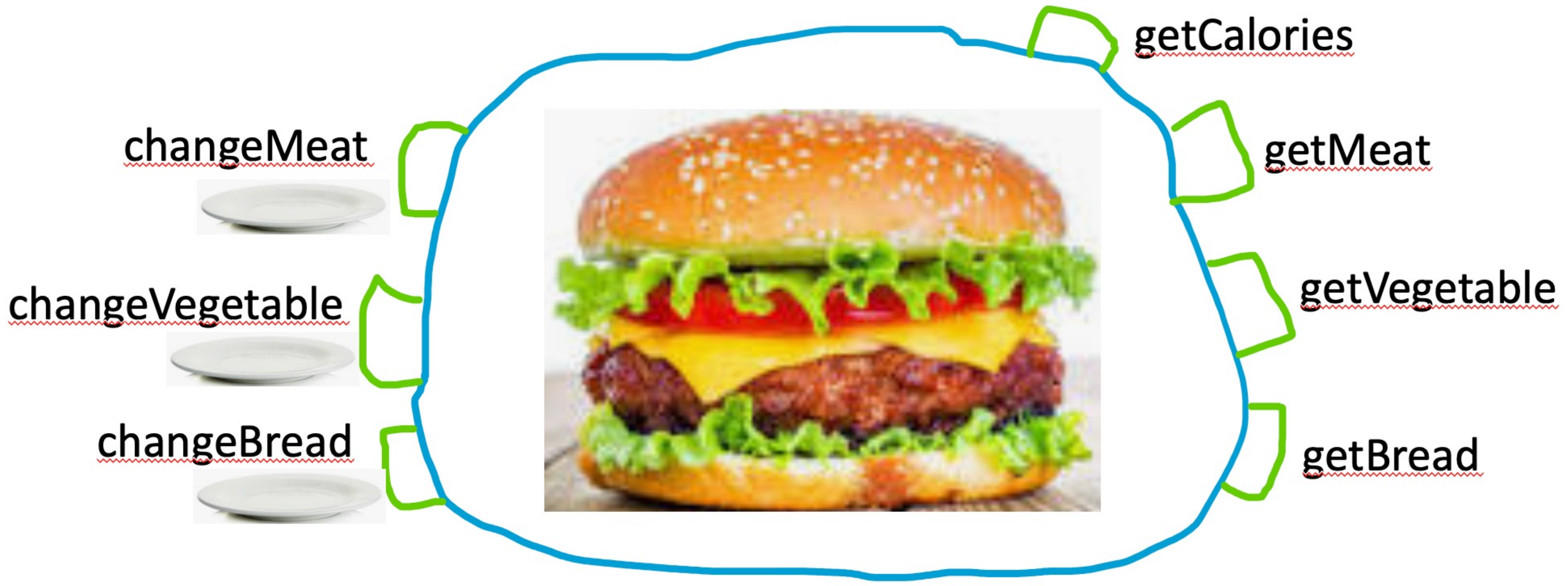
Object Oriented Programming

an intelligent hamburger

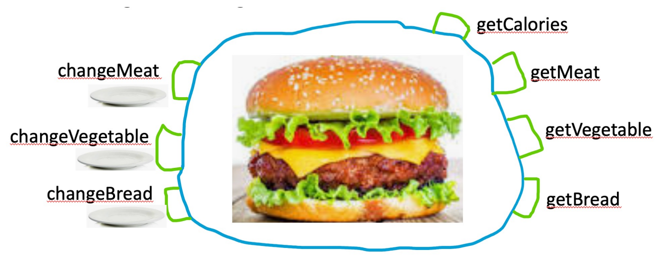


Data of an intelligent hamburger

- bread-, vegetable- and meat- layer.



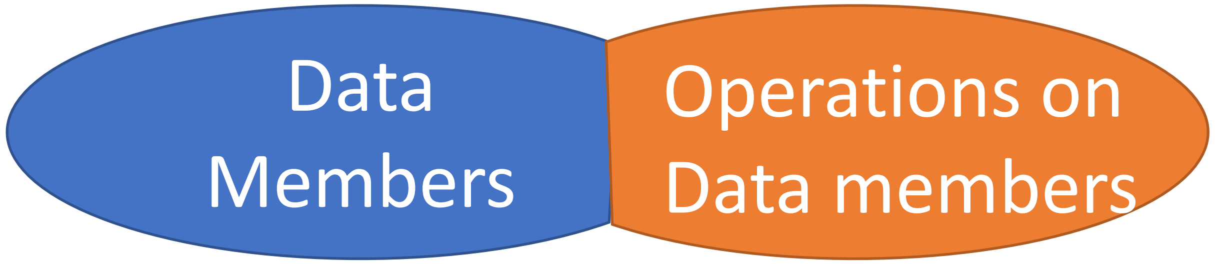
Operations for an intelligent hamburger



- 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.

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
```

Data members: II

```
string *values; //a string to represent the layout of data,
//mixed with possible empty strings.
 //Use array to access each element in const time.
 bool *bShown;
 //an array of boolean to indicate which element of
 //array values is shown or not.
 //If bShown[i] is true, then values[i] is shown,
 //otherwise values[i] is not shown,
 //where 0 <= i and i < numSlots.
```

Operations for data in Memory Game

```
public: //public method member, any class can use these methods
    MemoryGame();
      //default constructor, with 3 pairs of random integers in
      //range [0, 999], placed in 8 blocks (two blocks are empty).
    MemoryGame(int numPairs, int numSlots);
      //Place numPairs pairs of random integers in range [0, 999]
      //in numSlots space, need numPairs > 0, numSlots > 0, and
      //numSlots >= 2 * numPairs
```

Operations for data in Memory Game: II

```
MemoryGame(string *words, int size, int numSlots);
   //instead of randomly generated integers,
   //use words as data

~MemoryGame();
   //release dynamic allocated memory applied for
   //data members
```

Operations for MemoryGame class: III

```
void display() const;
//display array values, if bShown[i] is true,
//then values[i] is displayed, where i is the index.

void randomize();
//randomize the layout of elements in values
```

Operations for MemoryGame class: IV

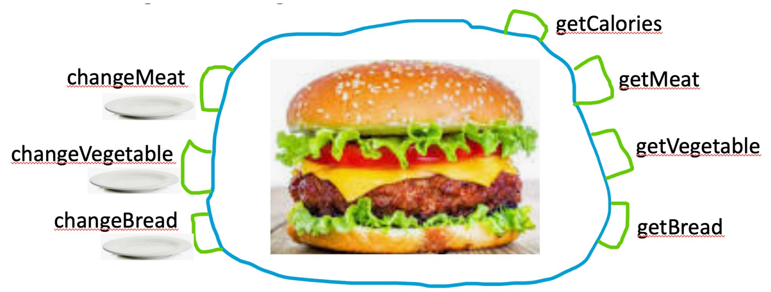
```
int input() const;
//input an int that is a valid index and
//the corresponding element of values is not shown yet.
//That is, the input i is in [0, numSlots) and
//bShown[i] is false.

void play(); //play the game
```

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.

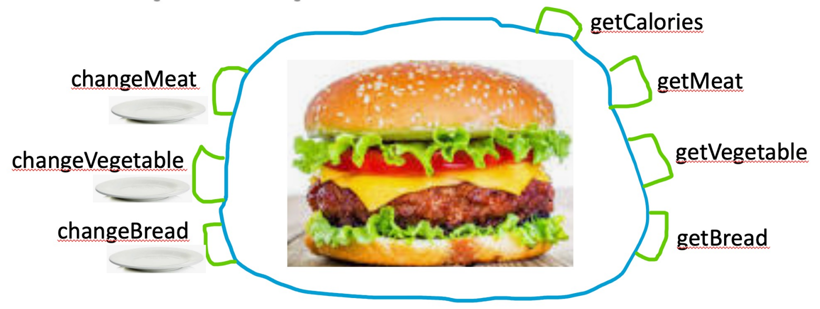
Constructors of Intelligent Hamburger

- The default constructor is a hamburger maker 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.
- Then a constructor adds operations (method members) to make the hamburger object intelligent.



Constructors of Intelligent Hamburger: II

- A non-default constructor takes parameters to "individualize" an hamburger. Say, one might like chicken instead of beef.
- A constructor creates a hamburger with those 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, …).
  - Operations are like buttons.



Task A: define constructors and destructors

- In MemoryGame.cpp
- Data members can be accessed without being passed as parameters

```
#include "MemoryGame.hpp"
//TODO: include other libraries
MemoryGame::MemoryGame() { //:: scope operator
        //initialize data members
        …
}
```

Task A: work on a non-default constructor first

```
MemoryGame::MemoryGame() {
}

MemoryGame::MemoryGame(int numPairs, int numSlots) {
}

MemoryGame::MemoryGame(string* words, int size, int numSlots) {
}
```

Task A: work on a non-default constructor first: II

```
MemoryGame::MemoryGame(int numPairs, int numSlots) {
    //TODO: check whether formal parameters are valid or not,
    //          if not, change them to be valid

    //TODO: set data members to by formal parameters

    //TODO: set values to be an array of strings with validated
    //numSlots elements
}
```

To be continued

Task A: work on a non-default constructor first: II

```
MemoryGame::MemoryGame(int numPairs, int numSlots) {
    //TODO: Generate numPairs random integers in [0, 999].
    //Convert numbers to strings,
    //put in pairs to the first (2*numPairs) slots of array values.

    //TODO: Set the rest elements of values to be "".
    //Set bShown to be an array of bool with numSlots elements

    //Set each element of bShown to be false.
}
```

continued

Do not forget

- Convert an int to a string using to_string function from std:: namespace.
  - Data member values is an array of strings.
- Release dynamically allocated memory and avoid dangling pointer problem in destructor.

Task A: work on non-default constructor first

```
MemoryGame::MemoryGame() : MemoryGame(3, 8) {
}
MemoryGame::MemoryGame(int numPairs, int numSlots) {
        … //code omitted
}
```

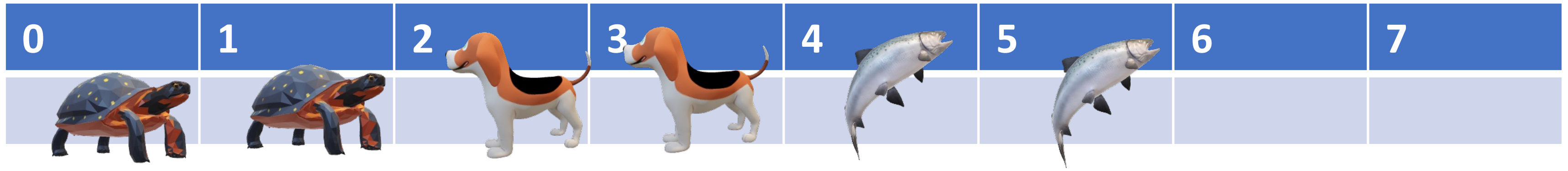
- Hamburger(string breadLayer, string meatLayer, string vegLayer)
- A default (or typical) hamburger is one with wheat bread, beef, lettuce and onion.

pseudocode

  - Hamburger(“wheat Bread”, “beef”, “lettuce and onion”)

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.



- 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
"383"	"383"	"886"	"886"	"777"	"777"	""	""

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

0	1	2	3	4	5	6	7
"383"	"383"	"886"	"886"	"777"	""	""	"777"

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
“383”	“383”	“886”	“886”	“777”	“”	“”	“777”

- Suppose we pick up index 3, swap the elements indexed at 3 and 6.

0	1	2	3	4	5	6	7
“383”	“383”	“886”	“”	“777”	“”	“886”	“777”

randomize elements in this segment

- Continue until the segment to be randomized has only one element.

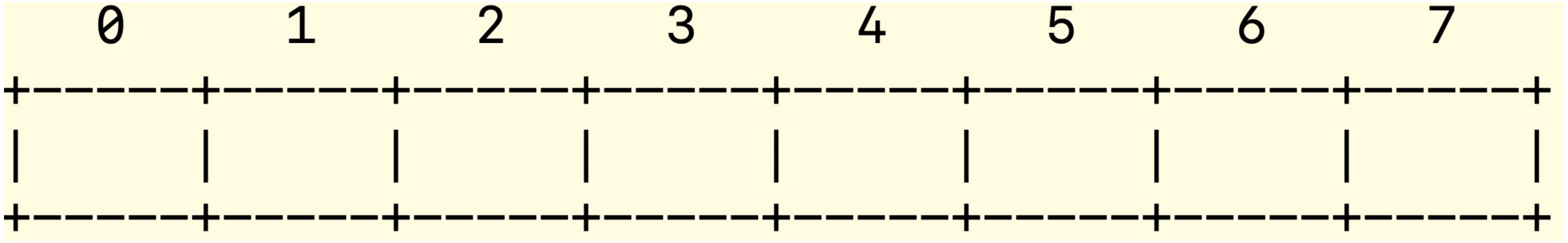
Task A is due on 4/7/24

- Define constructors and the destructor in MemoryGame.cpp.
- No main function can be included.
  - Tester scripts has main function as well.
  - In a C++ project, can have exactly one main function.
- Need to have **randomize** and **display** methods headers followed by {}.

```
void MemoryGame::randomize() {
}
void MemoryGame::display() const {
}
```

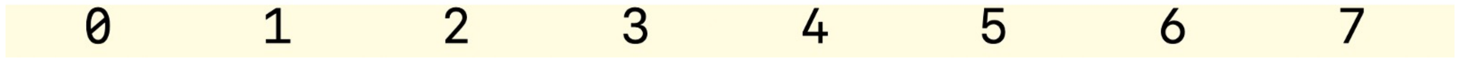
Method display of MemoryGame

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



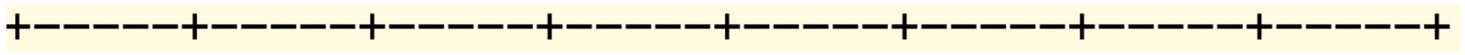
```
   0      1      2      3      4      5      6      7
+------+------+------+------+------+------+------+------+
|      |      |      |      |      |      |      |      |
+------+------+------+------+------+------+------+------+
```

- **Display indices (labels).**



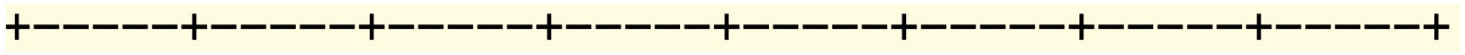
```
   0      1      2      3      4      5      6      7
```

- Display separate line.



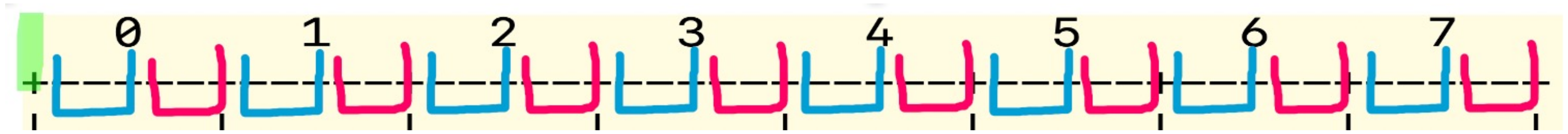
```
+------+------+------+------+------+------+------+------+
```

- **Display data if the corresponding bShown value is true, or display “”.**
- One more separate line.



```
+------+------+------+------+------+------+------+------+
```

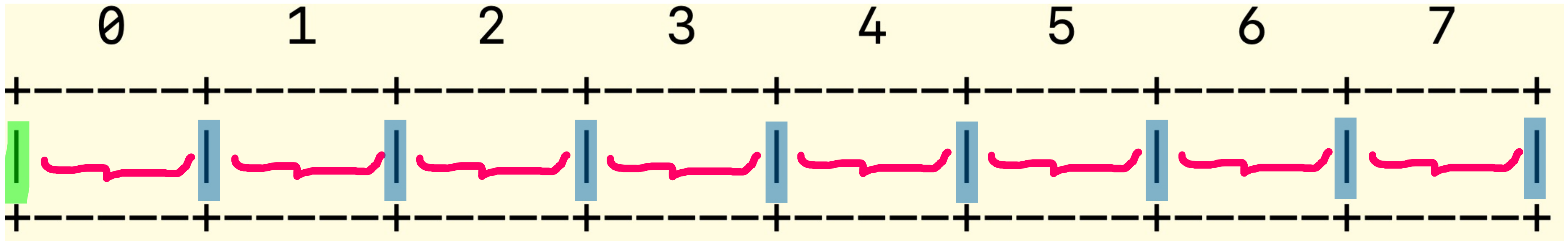
Method display : Display indices (labels).



Display 1 space, display index in 3 spaces, display 3 spaces

```
cout << " "; //the first space
for (int i = 0; i < numSlots; i++) {
     cout << setw(3) << i; //display index in 3 letter-width
     cout << setw(3) << " "; //display three spaces after index
}
cout << endl;
```

Method display of MemoryGame: III



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

- **Display data if the corresponding bShown item is true, or display empty string "".**

```
if (bShown[i]) //index i is in 0 <= i < numSlots
   cout << setw(5) << values[i]; //display ith item in array values
else cout << setw(5) << ""; //display empty string
```

Define input method

- int MemoryGame::input() const
- Keep on entering an integer from console until it is a valid index and is not yet flipped. Return the input.
- What means a valid index?
- What means a card is not flipped yet?

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 (values may change).

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

Define play method: II

- Key is to manipulate ***bShown*** array.
- Call **randomize** method.
- Flip cards until all matched pairs are found.
- Besides ***bShown***, use the following variables.
  - ***index***: the index of the current card being flipped
  - ***round***: number of rounds to find all matched pairs
  - ***pairsFound***: number of matched pairs found so far
  - ***first***: index of the first card flipped in a round.

Outline of play function

```
Call randomize method
Set variable pairsFound to be zero.
Set variable round to be zero.
As long as pairsFound < numPairs
Begin
        increase round by 1
        Choose a valid index whose cell is not displayed yet.
        flip a card (what to do in a flip, see next slide)
end
Report round taken to find all matched pairs.
```

What do we do in each flip?

```
if (it is the first flip)
begin
    Set the corresponding ***index*** of ***bShown*** to be true.
    Save the chosen ***index*** to variable ***first***.
end
//to be continued in the next slide
```

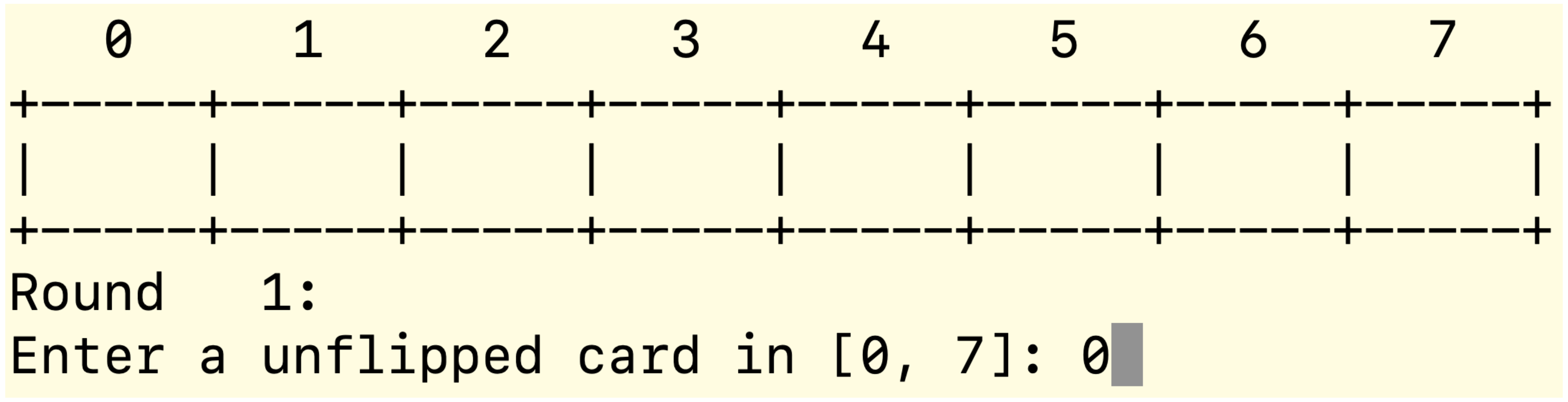
How do we know this is the first or the second flip?

What do we do in each flip? II

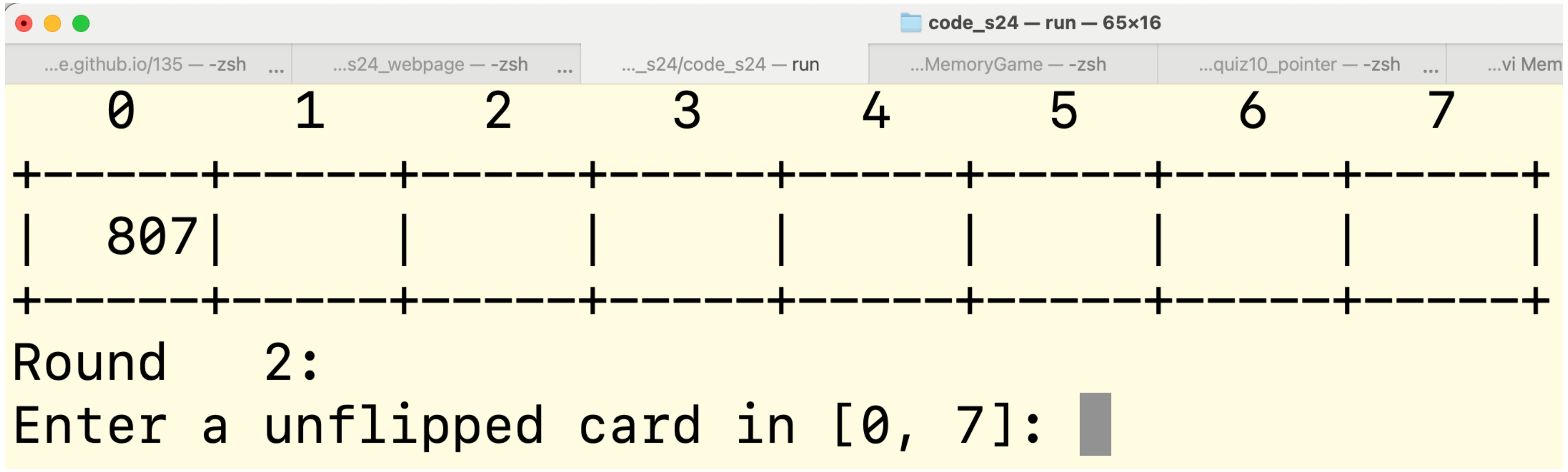
```
//continue from previous slide
else begin //this is the second flip in a round
     if the second flip matches the first flip and they are not empty string,
          set element of bShown to be true, increase pairsFound by 1,
     else set element of bShown at index first to be false.
    end
Display the layout.
~~Increase round by 1~~ //not needed, already increase in the beginning of loop
```

Optional improvement: each round is displayed in the top of a screen

- Press enter key.



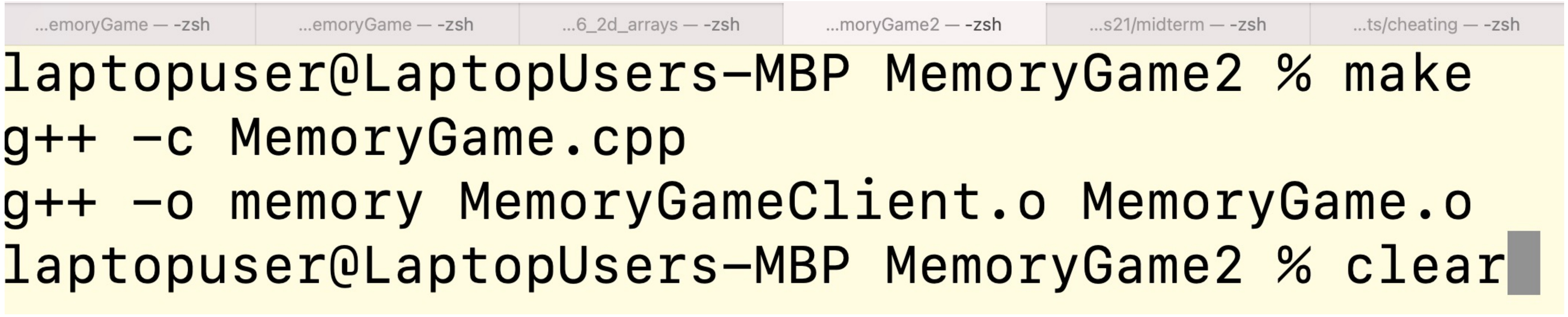
```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|     |     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Round   1:
Enter a unflipped card in [0, 7]: 0
```



```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|  807|     |     |     |     |     |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
Round   2:
Enter a unflipped card in [0, 7]: 
```

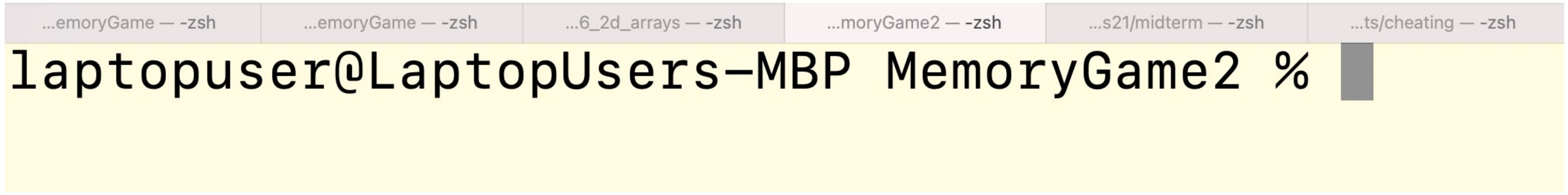
use clear method from Linux to flip screen

- Run commands in Linux, their outputs are shown in the screen.



```
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 clear command in C++ (optional)

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"); //call clear command
```

Reference:

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

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

How to test your code

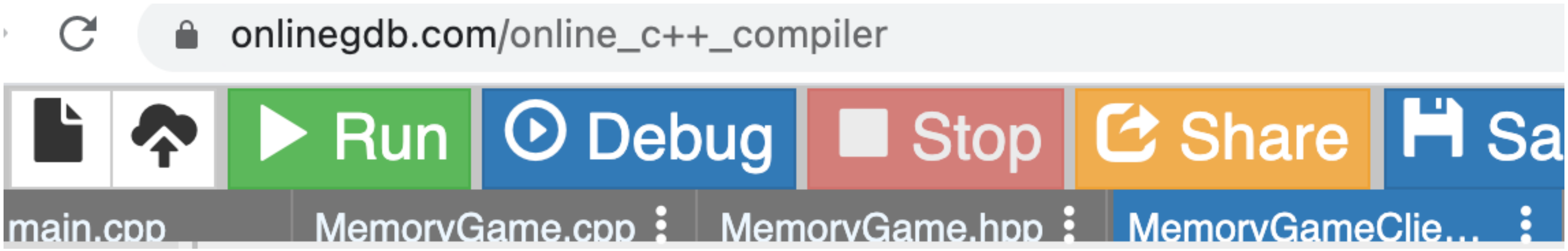
- Download the follow files from blackboard. Put in one directory.
  - MemoryGame.hpp (no modification is needed),
  - MemoryGame.cpp
  - MemoryGameClient.cpp (no modification is needed)
  - Makefile
- Under terminal, type in make and return key.
- Run ./run with return key.

Run code in onlinegdb C++

- Remove all the code in the main method attached.
  - There can be only one main function each C++ project
- 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 constructors and methods declared in MemoryGame.hpp.
    - It is like a factory branch to produce MemoryGame objects.
  - MemoryGameClient.cpp: test MemoryGame objects. This is like a Quality Analysis branch to test products.

Run projects in onlinegdb C++: II

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


onlinegdb.com/online_c++_compiler
Run
Debug
Stop
Share
Sa
main.cpp
MemoryGame.cpp
MemoryGame.hpp
MemoryGameClie...