Shape Sort Project

Tong Yi

Implement ball sort game in `https://play.google.com/store/apps/details?id=ball.sort.puzzle.color.sorting.bubble.games&hl=en_US`. We use colored shapes, not just balls.

Warning:

1. This is copyrighted materials; you are not allowed to upload to the Internet.
2. Our project is different from similar products in Internet. We use shapes instead of balls.
   (a) Ask help only from teaching staff of this course.
   (b) Use solutions from ChatGPT or online tutoring websites like, but not limited to, chegg.com, violates academic integrity and is not allowed.

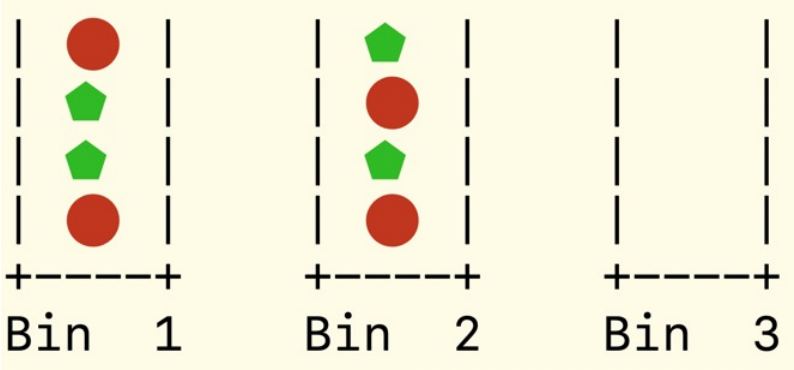
1 Rules

1. A SortGame object has several bins holding the same number of different shapes, randomly distributed in the bins.
2. For simplicity, assume all bins have the same capacity, that is, the maximum number of elements a bin can hold.
3. The number of each shape has the same value as capacity.
4. Provide some initially empty bins.
5. Rules for moving are listed as follows.
   (a) At any time, number of elements in a bin cannot exceed its capacity.
   (b) Moving out shapes from one bin to another until all elements with the same shape are put in some bin or there is no way to move.
      i. Only the top element of the move-out bin can be moved.
      ii. If there are consecutive elements of the same shape in the move-out bin, all those elements will be moved together. That is, all-same-shape-on-the-top elements are moved out or none.
      iii. The top element of the move-in bin must have exactly the same shape of the move-out bin unless the move-in bin is empty.
      iv. After adding the elements from the move-out bin, the elements in the move-in bin cannot exceed its capacity.

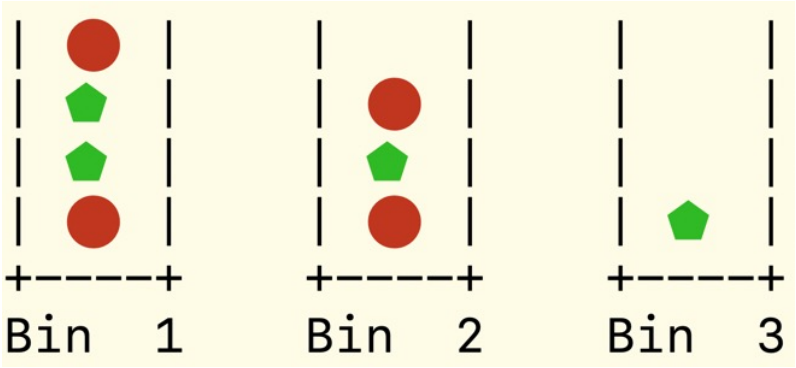
2 An example

Here is a sample run.

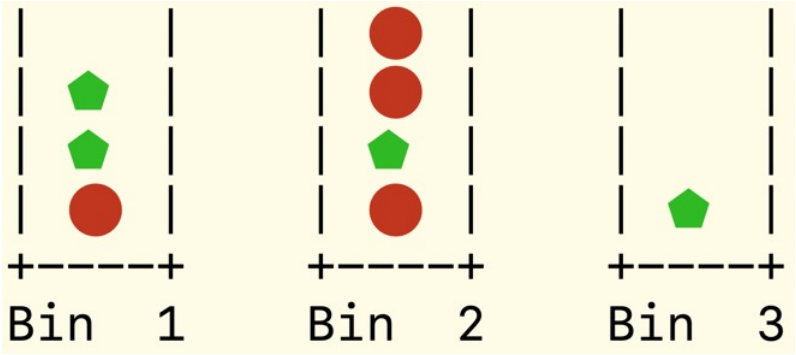
1. In the beginning,
   (a) Two bins with capacity 4 holding two randomly distributed shapes.
   (b) Each shape has 4 (same as capacity) elments.
   (c) One empty bin is provided.



2. As shown in the previous figure, Bin 3 is empty. Move the top element – a pentagon – from Bin 2 to Bin 3. After moving, we get the following distribution.



3. As shown in the previous figure, none of the bins is empty. Move can happen between bins whose top elements are of the same shape. Move the top element – a circle – from Bin 1 to Bin 2 since the latter is not full yet. After moving, we get the following distribution.



4. From the previous figure, only Bin 1 and Bin 3 share same-shape top elements.

   Move from Bin 1 to Bin 3. Note that all the top elements with the same shape in the move-out bin – in this example, there are two pentagons – are moved to the move-in bin. And we get the following distribution.

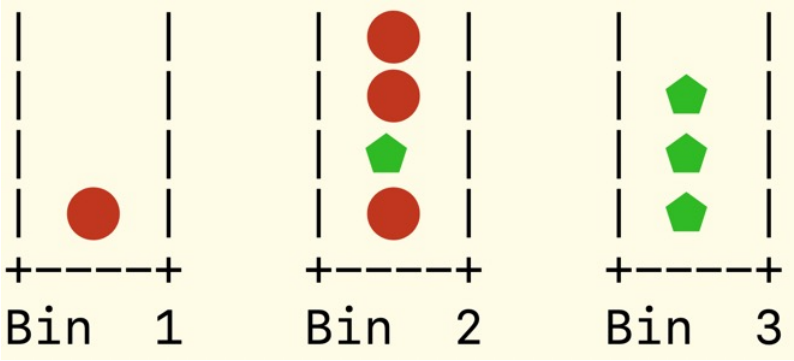
   (a) Warning: if we move from Bin 3 to Bin 1 from the above figure, and suppose we move two circles from Bin 2 to Bin 3.

   (b) Then only Bin 1 and Bin 2 share the same-shape (say pentagon) top elements.

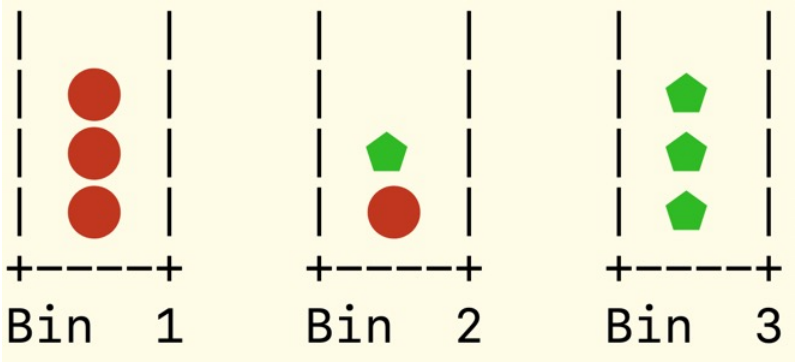
      i. However, Bin 1 is full and we cannot move from Bin 2 to Bin 1.

      ii. At the same time, Bin 1 has three pentagons but Bin 2 has only two empty slots left. Those three pentagons need to be move together and cannot be separated.

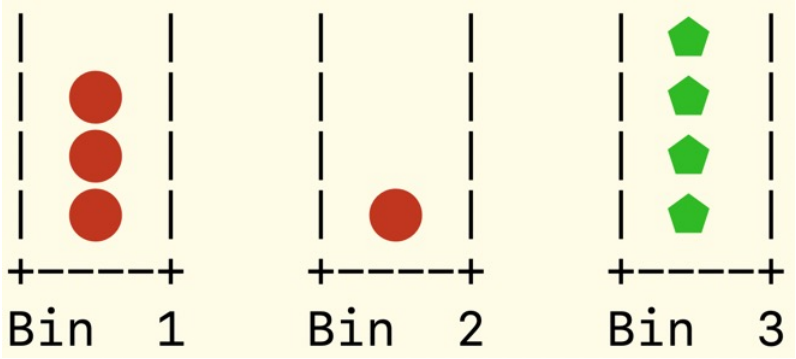
   (c) As a result, there is no more eligible move available at this point and we would fail the game.



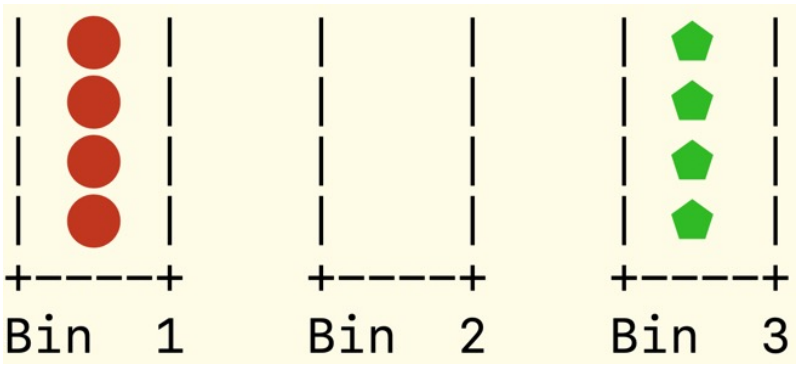
5. From the above figure, move two top circles from Bin 2 to Bin 1. We get the following distribution.



6. From the above figure, move the top element – a pentagon – from Bin 2 to Bin 3. We get the following distribution.



7. From the above figure, move the top circle from Bin 2 to Bin 1.

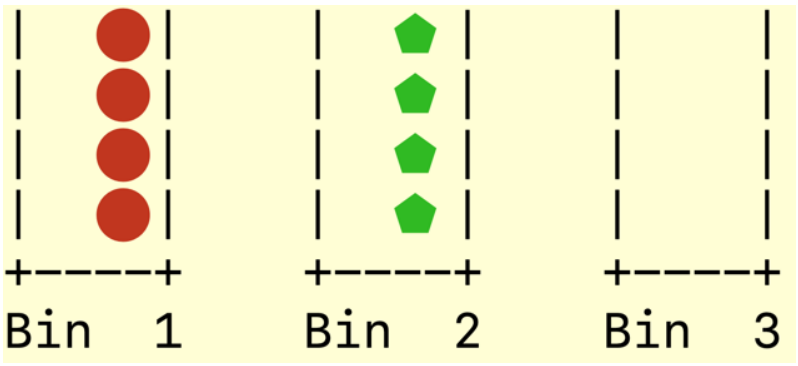


And the game runs successfully.

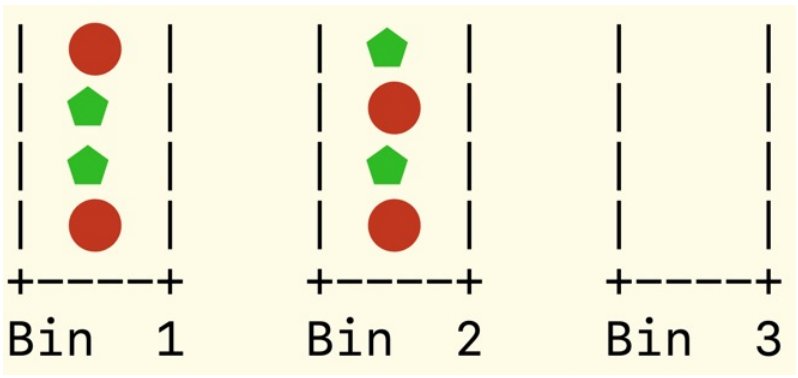
2.1 Intuitive Ideas for Steps

Suppose `numDiffElms` is 2 and `capacity` is 4. And `numEmptyBins` is 1.

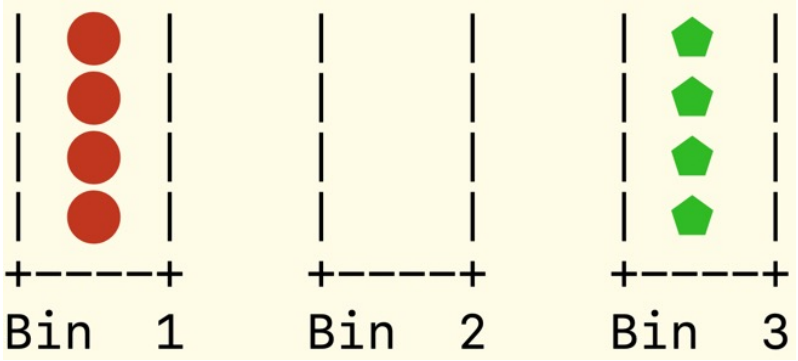
1. Start with sort-already (that is, not-randomized) data. For example, start with two bins that can hold at most four elements, each with a unique shape.



2. Randomize the elements in those `numDiffElms` bins. Suppose here is a randomized result.



3. After moving the shapes following the rules in Section 1, the shapes are sorted – but these sorted shapes can be in any bin.



3 Files of the Project

We use Object-oriented Programming approach.

1. Create directory `sortGame` to hold codes of the project **if** you have not done so. Said differently, you only need to run the following command once.

   ```
   mkdir sortGame
   ```

2. Move to the above directory.

   ```
   cd sortGame
   ```

3. Create `SortGame.hpp` with the following contents. **Warning:** do not write `SortGame.hpp` as `sortGame.hpp`. C++ is a case-sensitive language.

   `SortGame.hpp` is the header file of SortGame class that **declares** data members and operations (aka methods) on those data members.

```
#ifndef SortGame_H
#define SortGame_H

#include <vector>
#include <string>
class SortGame {
public:
    SortGame(); //2 bins, capacity 4 (each bin has at most 4 elements), 1 empty bin
    SortGame(int numDiffElms, int capacity, int emptyBins);

    //Normally in a header file (ended by .hpp in C++),
    //we do not use standard namespace by adding
    //using namespace std;
    //Reason: the header might be included in other files,
    //which may not like to use standard namespace.

    //WARNING: if not use -std=c++11 option to compile (that is, c++11 or higher),
    //then we cannot use >> as in the last two symbols of the following statement,
    //std::vector<std::vector<int>>,
    //need to use std::vector<std::vector<int> >
```

```
21      SortGame(std::vector<std::vector<int>> binData, int emptyBins);
22
23      void randomize();
24      void display() const;
25
26      //WARNING: a vector is passed by value by default.
27      //To pass the parameter of vector type as reference,
28      //must add & after the parameter.
29      bool move(std::vector<std::vector<int>>& shapesInBins, int& numBinsFinished);
30
31      void play();
32
33  private:
34      std::vector<std::vector<int>> bins;
35      int numDiffElms;
36      int numEmptyBins;
37      int capacity;
38  };
39  #endif
```

4. Your task is to implement `SortGame.cpp`, which **defines** constructors and methods declared in SortGame.hpp.

   (a) Note that, in `SortGame.hpp`, data members are declared but not yet initialized. The data members are initialized in constructors.

   (b) Similarly, constructors and methods are declared (have function header) in `SortGame.hpp` but not defined (no function body).

   (c) **Warning:** do NOT put main function in `SortGame.cpp`.

4 Data Members in SortGame.hpp

The details of data members, constructors and methods in SortGame class of the game are discussed as follows.

1. Data member `numDiffElms` is an integer representing the number of different shapes. In the above example, `numDiffElms` is 2. One shape is a red circle, the other is a green pentagon.

2. Data member `capacity` is an integer representing the maximum number of element each bin can hold. In the previous example, `capacity` is 4.

3. Data member `numEmptyBins` is an integer representing the number of empty bins. In the previous example, `numEmptyBins` is 1.

4. Data member `bins` of type `std::vector<std::vector<int>>` is a two dimensional array of integers and holds the values of different bins (including the initially empty ones).

(a) A vector is a one-dimensional array that can grow or shrink. It is a template class, documentation can be found at `https://cplusplus.com/reference/vector/vector/`.

(b) To use vector, need to include the library.

`include <vector>`

i. If you do not use standard namespace std, then need to add `std::` before vector.

ii. Example: declare a vector as an array of integers with 4 elements. Each element is initialized to be 1.

```
#include <iostream>
#include <vector>
#include <string>
//using namespace std;

int main() {
    std::vector<int> aBin = {1, 1, 1, 1};
        //aBin is a vector of integers with elements 1, 1, 1, 1.

    for (int i = 0; i < aBin.size(); i++)
        std::cout << aBin[i] << std::endl;
    return 0;
}
```

(c) Each bin is represented by a vector of integers, similar to a one-dimensional array of integers. A bin may be empty.

(d) In the previous example, data member `bins` has three bins:

i. The first bin is a vector with elements 1, 2, 2, 1.

ii. The second bin is a vector with elements 1, 2, 1, 2.

iii. The third bin is an empty vector.

(e) When displaying, map integer 1 to red circle and integer 2 to green pentagon. More mapping details are shown in `display` method.

You may think `bins` in the previous example has three rows, the first row has four (same value as capacity) columns, so is the second row. The third row is empty in the beginning.

Each row can grow and shrink, each has at most four columns.

Note that data member `capacity` does not suppose a limit on the size of the vector. When this capacity is exhausted and more is needed, it is automatically expanded by the container (reallocating it storage space).

However, for this game, we need to make sure that no bin can have more `capacity` elements at any time. When there is `capacity` elments in a bin, we stop adding elements to that bin.

5 Task A: Define constructors in SortGame.cpp

The purpose of constructor is to initialize data members. A class may have multiple constructors. Different constructors have different parameter lists. Each constructor has exactly the same name as class, no return type, not even void.

5.1 The default constructor `SortGame()`

The default constructor does not take any parameter. It does the following:

1. Set data members `numDiffElms` to be 2.
2. Set data member `capacity` to be 4.
3. Set data member `numEmptyBins` to be 1.

   Warning: the following code is wrong. `int` before `numDiffElms` means to the varialbe is a local variable for constructor `SortGame`, but not data member `numDiffElms`.

```
SortGame::SortGame() {
    int numDiffElms = 2;
    ... //omit other code
}
```

   Correct way:

```
SortGame::SortGame() {
    numDiffElms = 2;
    ... //omit other code
}
```

4. You may use the hints from the following code to initialize data member `bins`.

```
//for each shape, do the following:
for (int i = 0; i < ?; i++) { //TODO: fill in ?
    std::vector<int> row;
       //row is a vector of ints created by default constructor of vector,
       //that is, row is an empty vector
       //You may think a row as a bin in our application.

    for (int j = 0; j < ??; j++) { //TODO: fill in ??
        row.push_back(???); //TODO: fill in ???
        //hints: the first bin has capacity elements,
        //each element is 1
        //the second bin has capacity elements,
        //each element is 2,
        //and so on.
    }

    //add the one-dimensional array row to bins
    bins.push_back(row);
}

//add empty bins to bins
for (int i = 0; i < ????; i++) { //TODO: ????
    //TODO: create an empty vector object row
```

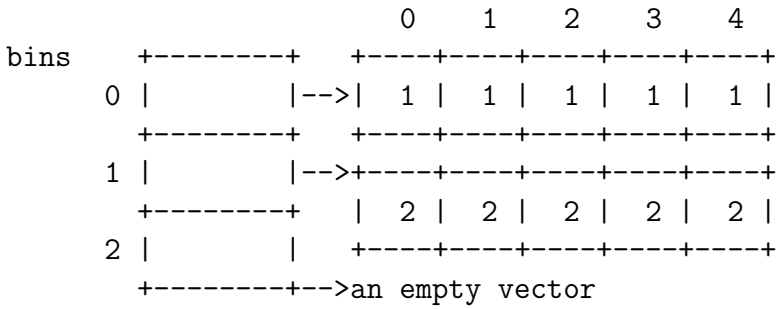
```
24
25      //TODO: Use push_back method to add row to bins
26
27  }
```

5. In Task A, we do not randomize the placement of numbers yet.

5.2 A nondefault constructor `SortGame(int numDiffElms, int capacity, int numEmptyBins)`

1. If given parameter `numDiffElms` is smaller than 1, reset it to be 1.
2. If given parameter `capacity` is smaller than 2, reset it to be 2.
3. If given parameter `numEmptyBins` is smaller than 1, reset it to be 1.
4. Now given parameters are correct, use them to set the corresponding data members. Note that if a formal parameter has exactly the same name as a data member, we need to put `this->` before the data member, where `this` is a pointer to the current object.
5. In Task A, we do not randomize the placement of numbers yet.

   Suppose we call `SortGame(2, 5, 1)`, the layout of data `bins` before calling `randomize` method is as follows.



5.3 A nondefault constructor `SortGame(std::vector<std::vector<int>> binData, int numEmptyBins)`

1. Call `size` method of `binData` – which returns the number of rows in `binData` – to initialize data members `numDiffElms`.
2. Initialize data member `capacity` to be the number of elements in a row of `binData`. Assume that rows of `binData` have exactly the same number of elements, without loss of generality, we can use the number of elements in the first row, whose row index is 0, to initialize data member `capacity`.

   The first row of `binData` is `binData[0]`.

   The number of elements in the first row of `binData` is found by calling `size` method on the first row.
3. It remains to add each row of `binData` to data members `bins`.

4. Do not forget to add `numEmptyBins` empty bins to data member `bins`.

5. Note that data in `binData` is randomized already, for example, the first row has elements 1, 2, 1, 1, the second row has elements 2, 2, 1, 2, there is no need to call `randomize` method in this constructor.

You may notice that there are a lot of common codes among those constructors. A better way is to define `SortGame(int numDiffElms, int capacity, int numEmptyBins)`. Then use constructor delegate to define `SortGame()` and `SortGame(std::vector<std::vector<int>> binData, int numEmptyBins)`.

No need to define destructor in this project since we did not dynamically allocate memories for data members.

5.4 Finish Task A

1. Define constructors in `SortGame.cpp`.

```cpp
#include "SortGame.hpp"
#include <iostream> //cout
#include <iomanip> //setw
#include <cstdlib> //rand
#include <ctime> //time
#include <algorithm> //swap

//TODO: fill in ?, ??, and ??? in the parentheses.
//Hint: what are the values of numDiffElms and capacity for a default SortGame
   object?
//Question: after calling SortGame(?, ??, ???) to create a SortGame object with
//? different shapes,
//each bin holds at most ?? elements,
//There are ??? empty bins.
SortGame::SortGame() : SortGame(?, ??, ???) {
    //No more code is needed
}

SortGame::SortGame(int numDiffElms, int capacity, int numEmptyBins) {
    //TODO: If given parameter numDiffElms is smaller than 1,
    //reset it to be 1.


    //TODO: If given parameter capacity is smaller than 2,
    //reset it to be 2.


    //TODO: If given parameter numEmptyBins is smaller than 1,
    //reset it to be 1.


    //Now given parameters are correct,
    //use them to set the corresponding data members.
```

```
33      //Note that if a formal parameter has exactly
34      //the same name as a data member,
35      //we need to put this-> before the data member,
36      //where this is a pointer to the current object.
37
38      //TODO: use formal parameter numDiffElms to set data member numDiffElms
39
40
41      //TODO: use formal parameter capacity to set data member capacity
42
43
44      //TODO: use formal parameter numEmptyBins to set data member numEmptyBins
45
46
47      //initialize data member bins
48
49      //TODO: add numDiffElms rows -- each row is a bin -- to bins,
50      //      for ith bin (aka, row) ranging from 0 to numDiffElms - 1,
51      //      put capacity many ints with value (i+1) to the ith bin
52      //
53      //for i in [0, numDiffElms):
54      //begin
55      //   instantiate an empty bin, call it row
56      //   push capacity many ints with value (i+1) to row,
57      //   push row back to data members bins
58      //end
59
60
61
62
63
64
65      //TODO: add numEmptyBins rows -- each row is a bin -- to bins.
66      //Step is similar to the previous step,
67      //except without the step of
68      //pushing capacity many ints with value (i+1) to row.
69      //since these bins are empty.
70
71
72
73
74
75  }
76
77  //TODO: fill in ... in the parentheses. These values are different.
78  //Hint: what are the values of numDiffElms and capacity
```

```
//      for a SortGame object with binData?
//For example, suppose binData is {{1, 2, 1, 1}, {2, 2, 1, 2}},
//what is numDiffElms and capacity?
SortGame::SortGame(std::vector<std::vector<int>> binData, int numEmptyBins)
    : SortGame(..., ..., ...) {
    //TODO: initialize data members bins from binData

}
```

2. Test codes locally.

   (a) Comment private: line in `SortGame.hpp` as `//private:`. This is for debug purpose.

   (b) Edit main.cpp as follows.

```
#include <iostream>
#include <vector>
#include "SortGame.hpp"
//g++ -std=c++11 SortGame.cpp main.cpp
//test default constructor using
//./a.out A or ./a.out 'A'
//./a.out B or ./a.out 'B'
//./a.out C or ./a.out 'C'

int main(int argc, const char *argv[]) {
    if (argc != 2) {
       std::cout << "Need 'A'-'C' in parameters" << std::endl;
       return -1;
    }

    //unit-testing for constructors and the destructor
    char type = *argv[1];
    std::string prompt;
    SortGame *game;
    int** arr;
    if (type == 'A') {
       prompt = "default constructor,";
       game = new SortGame;
    }
    else if (type == 'B') {
        prompt = "SortGame game(6, 3, 2);";
        game = new SortGame(6, 3, 2);
    }
    else if (type == 'C') {
        prompt = "SortGame game(vector<vector<int>>, 2);";
        //std::vector<std::vector<int>> binData = { {3, 2, 2, 3}, {1, 1, 2,
   3}, {1, 1, 3, 2} }; //cannot omit std:: before vector if not using
   namespace std;
```

```
32          //game = new SortGame(binData, 1);
33          //Use the following example to test 6 * 5 two-dimensional array.
34          std::vector<std::vector<int>> binData = { {6, 2, 5, 3, 4}, {3, 2, 1, 4,
     4}, {1, 4, 2, 3, 5}, {1, 5, 6, 3, 2}, {6, 5, 6, 1, 2}, {3, 4, 5, 6, 1} };
35          game = new SortGame(binData, 2);
36      }
37
38      std::cout << "After " << prompt
39           << " data member numDiffElms is " << game->numDiffElms << std::endl;
40      std::cout << "After " << prompt
41           << " data member capacity is " << game->capacity << std::endl;
42      std::cout << "After " << prompt
43           << " data member numEmptyBins is " << game->numEmptyBins << std::endl
    ;
44      std::cout << "After " << prompt
45           << " the first " << game->numDiffElms << " bins are " << std::endl;
46
47      //for (int i = 0; i < game->bins.size(); i++) { //modified,
48      for (int i = 0; i < game->numDiffElms; i++) { //just show the bins with
    data
49          for (int j = 0; j < game->bins[i].size(); j++) {
50              std::cout << game->bins[i][j];
51              if (j < game->bins[i].size()-1) //skip the last ,
52                  std::cout << ",";
53          }
54          std::cout << std::endl;
55      }
56
57      //(1) index of non-empty bins is from 0 to game->numDiffElms -1
58      //(2) index of empty bins is from game->numDiffElms to game->bins.size().
59      //(3) For example, if bins.size() is 3 -- three bins,
60      //    and suppose numDiffElms is 2,
61      //    then the first two bins -- indices 0 and 1 --
62      //    are not empty, and empty bin runs from 2 to 3 (not included)
63      int actualNumEmptyBins = 0;
64      for (int i = game->numDiffElms; i < game->bins.size(); i++)
65          if (game->bins[i].size() != 0) {
66              std::cout << "Error: the size of empty bin should be 0" << std::
    endl;
67              return -1;
68          }
69          else actualNumEmptyBins++;
70
71      std::cout << "After " << prompt << " ";
72      if (actualNumEmptyBins == 1)
73          std::cout << "there is " << actualNumEmptyBins << " empty bin" << std::
```

```
    endl;
74   else std::cout <<"there are " << actualNumEmptyBins << " empty bins" << std
    ::endl;
75
76   delete game;
77   game = nullptr;
78
79   return 0;
80 }
```

(c) Run the following command to compile main.cpp and SortGame.cpp.

`g++ -std=c++11 main.cpp SortGame.cpp`

(d) If there is no compilation errors, run the following command.

`./a.out A`

(e) You should be able see something like the following.

```
1 After default constructor, data member numDiffElms is 2
2 After default constructor, data member capacity is 4
3 After default constructor, data member numEmptyBins is 1
4 After default constructor, the first 2 bins are
5 1,1,1,1
6 2,2,2,2
7 After default constructor, there is 1 empty bin
```

(f) Test non-default construtor `SortGame(int numDiffElms, int capacity, int numEmptyBins)` by using

`./a.out B`

You should see the following output.

```
1  After SortGame game(6, 3, 2); data member numDiffElms is 6
2  After SortGame game(6, 3, 2); data member capacity is 3
3  After SortGame game(6, 3, 2); data member numEmptyBins is 2
4  After SortGame game(6, 3, 2); the first 6 bins are
5  1,1,1
6  2,2,2
7  3,3,3
8  4,4,4
9  5,5,5
10 6,6,6
11 After SortGame game(6, 3, 2); there are 2 empty bins
```

(g) Test non-default construtor

`SortGame(std::vector<std::vector<int>> binData, int numEmptyBins)` by using

`./a.out C`

You should see the following output.

```
After SortGame game(vector<vector<int>>, 2); data member numDiffElms is 6
After SortGame game(vector<vector<int>>, 2); data member capacity is 5
After SortGame game(vector<vector<int>>, 2); data member numEmptyBins is 2
After SortGame game(vector<vector<int>>, 2); the first 6 bins are
6,2,5,3,4
3,2,1,4,4
1,4,2,3,5
1,5,6,3,2
6,5,6,1,2
3,4,5,6,1
After SortGame game(vector<vector<int>>, 2); there are 2 empty bins
```

3. Or you can test the code in `https://www.onlinegdb.com/online_c++_compiler`.

   Upload main.cpp, SortGame.hpp (comment private: line) and SortGame.cpp to onlinegdb. In the textbox right to **Command line arguments:**, enter A or B or C.

4. If the code runs correctly in a local computer, upload `SortGame.cpp` to gradescope.

5. Again, do not add `main` function in `SortGame.cpp`.

6 Task B: define randomize and display methods

In Task A, we write codes for constructors. However, the shapes in the default constructor and constructor `SortGame(int numDiffElms, int capacity, int numEmptyBins)` are not randomized yet.

Note that shapes in data members `bins` constructed from `SortGame(std::vector<std::vector<int>> binData, int emptyBins);` do not need to be randomized.

1. Initialize `numDiffElms` and `capacity` to be valid integers, representing number of rows and number of columns of a two-dimensional array, respectively.

2. Put `capacity` many 1's in the first bin (a one-dimensional array represented by a vector), and put `capacity` many 2's in the second bin, $\cdots$, and put `capacity` many `numDiffElms`'s to the (numDiffElms)th bin.

3. It remains to randomize the elements in the first `numDiffElms` rows and `capacity` columns of data member `bins`. This is done in method `randomize`.

6.1 Method randomize

You must **follow the steps** to randomize the layout of these integers, otherwise, your code cannot pass gradescope.

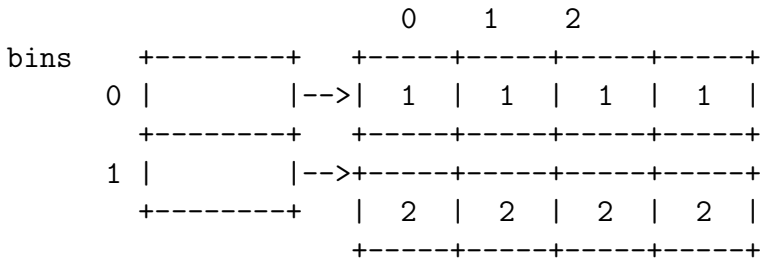
6.2 Approach 1: Randomize using one-dimensional intermediate vector

First we illustrate the idea with the following example.

6.2.1 An example of randomization

Suppose data member `numDiffElms` is set to be 2 and `capacity` is initialized to be 4. And `numEmptyBins` is initialized to be 1. However, we do not need to work with the empty bins in `randomize` method.

Then data member `bins` is laid out as follows, after running steps in constructors in Task A. We do not draw the empty vector since it is not used in randomization.



Note that `bins[0][0]`, `bins[0][1]`, `bins[0][2]`, and `bins[0][3]` have value 1, while `bins[1][0]`, `bins[1][1]`, `bins[1][2]`, and `bins[1][3]` have value 2. So `bins` as a 2-dimensional array created from vector can be TREATED as the following statically allocated 2-dimensional array.

The difference is, for a statically allocated 2-dimensional array, the number of columns must be a constant, however, for a dynamically allocated array or an array created by vector, its number of columns can be a variable.

bins

row index \ col index	0	1	2	3
0	1	1	1	1
1	2	2	2	2

Imagine the above elements are **linearized** as laying out elements from top row to the bottom row, for each row, move from left to right. (i, j) for row index i and column index j, where $0 \leq i <$ `numDiffElms` and $0 \leq j <$ `capacity`.

(row index, col index)	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(1, 0)	(1, 1)	(1, 2)	(1, 3)
	1	1	1	1	2	2	2	2

We can map (i, j) to $i *$ `capacity` $+ j$. In the above example, `capacity` is 4, so $(0, 1)$ is mapped to $0 * 4 + 1 = 1$ and $(1, 0)$ is mapped to $1 * 4 + 0 = 4$. Reason:

- BEFORE the FIRST element at row index i, there are $i *$ `capacity` integers.
- At ith row, there are j integers BEFORE the element at column index j.

So it is like we have a one-dimensional array as follows, where 1-d is a shortcut for one-dimensional and 2-d is for two-dimensional.

index in a 1-d array	0	1	2	3	4	5	6	7
(row index, col index) in a 2-d array	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(1, 0)	(1, 1)	(1, 2)	(1, 3)
	1	1	1	1	2	2	2	2

Next we will randomize the following one-dimensional array linearized from the original two-dimensional array.

index in one-dimensional array	0	1	2	3	4	5	6	7
	1	1	1	1	2	2	2	2

1. Let `currLastIdx` to be the current last index of the array. Initialize it to be `numDiffElms * capacity -1`. In the above example, `numDiffElms` is 2, `capacity` is 4, so the last index for the linearized (also called flattened) array is 2 * 4 - 1 = 7.

2. Choose a random index from 0 to `currLastIdx`, that is,

```
1 rand() % ( currLastIdx + 1 )
```

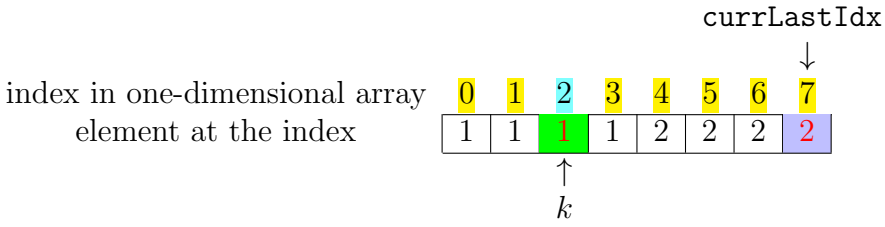
   Warning:

   - The denominator is `(currLastIdx +1)` instead of `currLastIdx`, otherwise, only indices from 0 to `currLastIdx - 1` are selected.
   - The parentheses around `currLastIdx + 1` **cannot** be omitted. Reason: remainder operator % has high precedence than operator +, so `rand() % currLastIdx + 1` runs `rand() % currLastIdx` first, then add 1 to that random number, so the chosen index is in `[1, currLastIdx]`, not the expected `[0, currLastIdx]`.

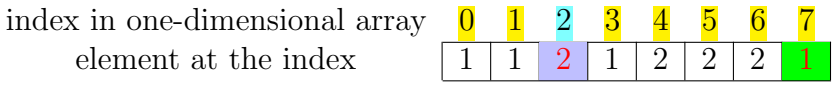
   Assume that 2 is selected. Save this index in variable k.

3. Swap the element indexed at k with the element indexed at current last index `currLastIdx`. Doing so would avoid to select that same element again in next round of randomization. In the above example, we get the following layout.

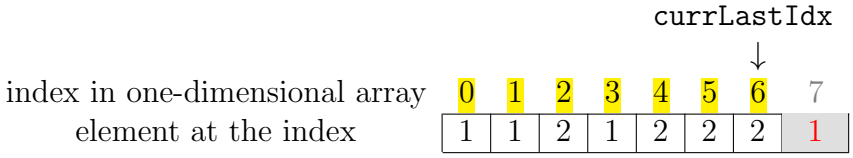
   Before swapping:



   After swapping:



4. Reduce `currLastIdx` by 1. The array looks as follows, as if the last element **were** truncated. So element 1, which is **indexed** at 7 after one randomization, will not be selected again.



5. Repeat Steps 2, 3, and 4 until `currLastIdx` is 0. That is, there is no more randomization is needed.

Here is a pseudocode.

```
//Suppose elements 1, ..., numDiffElms * capacity are saved in array temp,
//from the first index to the last index.

declare and initialize currLastIdx to be ...
while (currLastIdx > 0)
begin
    select a random integer in [0, currLastIdx], save in variable k.
    swap temp[k] with temp[currLastIdx]
    decrease currLastIdx by 1
end
```

6.2.2 Steps to randomize using one-dimensional intermediate vector

1. Instantiate an empty vector `vect` – or you can give it another proper name.
2. Put elements in data member `bins` with `numDiffElms` non-empty bins and each bin with `capacity` integers into the above empty vector `vect`. The order of adding elements (use `push_back` method from vector class) goes from the first row (row index 0) to the last row (row index `numDiffElms -1`), and for each row, from the leftmost element (column index 0) to the rightmost element (column index `capacity -1`).
3. Randomize elements in vector `vect` using steps listed in the previous example.
4. Put the elements in the vector `vect` backs to the `numDiffElms` non-empty bins, in the order from the first row to the last row, and from each row, from the left column to the right column.

6.3 Approach 2: Randomize directly on data member `bins`

In the previous approach, we use a one-dimensional array to save the data, randomize them, then copy the randomized data back to data member `bins`. In this approach, we randomize directly in two-dimensional array `bins`, without the need to use a one-dimensional array.

Label elements in data member `bins` from the top row to the last row; in the same row, from left to right. Label the top left element as 0, its right neighbor to be 1, and so on.

Map label k, where $0 \le k \le$ `numDiffElms` $*$ `capacity` -1, to row index k / `capacity` and column index k % `capacity`. For example, when `capacity` is 4 and k is 6, the corresponding row index is $6 / 4 = 1$ and the column index is $6 \% 4 = 2$.

bins

row index \ col index	0	1	2	3
0	1	1	1	1
1	2	2	2	2

labels of elements

row index \ col index	0	1	2	3
0	0	1	2	3
1	4	5	6	7

Next we will randomize elements in the two-dimensional array as follows.

1. Initialize `currLastIdx` to be _____ (you fill in the blank, this expression is related with `numDiffElms` and `capacity`). In our example, it is $2*4-1=7$, where 2 is the value of `numDiffElms` and 4 is the value of `capacity`.

2. Select a random integer in `[0, currLastIdx]`, where 0 in the first label and 7 is the last label. Suppose 2 is chosen.

3. Map label 2 back to row index `2 / capacity = 2 / 4 = 0` and column index `2 % capacity = 2 % 4 = 2` in the two-dimensional array.

4. Map `currLastIdx` 7 back to row index `7 / capacity = 7 / 4 = 1` and column index 7 % capacity = 7 % 4 = 3 in the two-dimensional array.

5. Swap elements at (0, 2) and (1, 3) in the original two-dimensional array `bins`.

   It is like element indexed at (0, 2) in the two-dimensional array is chosen, we swap it with the element at `currLastIdx` to avoid to choose it again.

   Before swapping:

   bins

row index	col index 0	1	2	3
0	1	1	1	1
1	2	2	2	2

   After swapping:

   bins

row index	col index 0	1	2	3
0	1	1	2	1
1	2	2	2	1

6. Reduce `currLastIdx` by 1.

7. Repeat Steps 2-6 until `currLastIdx` is reduced to be 0.

Here is a pseudocode.

```
declare and initialize currLastIdx to be ...
while (currLastIdx > 0)
begin
    select a random integer in [0, currLastIdx], save in variable k.
    swap bins[k / capacity][k % capacity] with bins[currLastIdx / capacity][currLastIdx
    % capacity]
    decrease currLastIdx by 1
end
```

6.4 Define `display` method

Suppose after randomize, data member `bins` has two bins,

bins

row index	col index 0	1	2	3
0	1	2	1	1
1	2	2	1	2

We would like to display the bins (also called rows) **vertically** from left to right and each bin is displayed **upside down**. That is, the element at column index 0 for a row is displayed at the bottom while the the element at column index `size of that bin -1` is displayed at the top.

Reason: we assume that the bins are opened from the back, so we can push an element to the end and pop an element from the end.

6.4.1 Display the first bin vertically, upside down

```
1
1
2
1
```

6.4.2 Elements in a bin with left-, right- and bottom-bounds for the bin displayed

Warning: must use spaces, cannot use `'\t'` (tab key) to replace spaces.

The number of spaces between each bin is 4.

No spaces after the last bin.

Do not use setw before bins[i][j], that is, `std:cout << std::setw(3) << bins[i][j];`, since later on we would need to map integer to a shape by replacing bins[i][j] by colorShapeMap[bins[i][j]-1], a shape represented by a unicode. However, `setw` does not apply to a unicode.

You can print two spaces before and one space after the unicode shape, followed by a vertical line `|`.

```
|  1 |
|  1 |
|  2 |
|  1 |
+----+
Bin  1
```

6.4.3 Display all bins from left to right with bounds

We would display all bins from left to right, as follows.

```
|  1 |    |  2 |    |    |
|  1 |    |  1 |    |    |
|  2 |    |  2 |    |    |
|  1 |    |  2 |    |    |
+----+    +----+    +----+
Bin  1    Bin  2    Bin  3
```

Consider capacity and the size of each bin. Capacity is the maximum number of elements a bin can hold, while size is the number of elements current residing in a bin. As an analog, a hotel may have 100 rooms, but only 20 are occupied. In that example, capacity is 100, and 20 is the current size. In any time, the size of a bin cannot be larger than its capacity.

Modify your code to handle the case when the number of elements in a bin does not equal to its capacity. For example, after moving the top two elements from Bin 1 to Bin 3, the elements in bins look as follows.

Note that the capacity for each bin is still 4, but the size of the first bin and the third bin is 2, and the size of the second bin is 4, where size is the number of elements in a bin.

```
|    |    |  2 |    |    |
|    |    |  1 |    |    |
|  2 |    |  2 |    |  1 |
|  1 |    |  2 |    |  1 |
+----+    +----+    +----+
Bin  1    Bin  2    Bin  3
```

Afterwards, map 1 to a red circle.  , map 2 to a green pentagon  , and so on.

Hints:

```
void SortGame::display() const {
    std::string shapes[] = {
       "\u2b24", //circle
       "\u2b1f", //pentagon
       "\u25fc", //medium square, can add color
       "\u272f", //star, can add color
       "\u2665", //heart
       "\u25b2", //triangle
    };
    int shapes_size = sizeof(shapes) / sizeof(shapes[0]);

    std::vector<std::string> colorShapeMap(numDiffElms);
    int colorCode = 31;
    //"\033[31m", a const char*, represents red text color
    //"\033[32m" represents green text color
    for (int i = 0; i < numDiffElms; i++) {
        colorShapeMap[i] = "\033[" + std::to_string(colorCode) + "m" + shapes[i %
   shapes_size] + "\033[0m";
        //restore to \033[0m, black color, after displaying a colored shape
        colorCode++;
    }


    //your code follows
    //TODO: replace statement
    //std::cout << bins[i][j];
    //by
    //std::cout << colorShapeMap[bins[i][j]-1];
}
```

Explanation: `"\033[31m"`, a const char*, represents red text color. `"\033[32m"` represents green text color, while `"\033[0m"` represents black text color.

`"\u2b24"`, where `\u` means unicode, represents a circle.

The following code print out a red circle and a green pentagon.

```
#include <iostream>
#include <string>
```

```
using namespace std;

int main() {
    cout << "\033[31m \u2b24" << endl; //red circle
    cout << "\033[32m \u2b1f" << endl; //green pentagon
    return 0;
}
```

So `"\033[31m \u2b24"` means a red circle. Since the number of shapes is limited and may be smaller than `numDiffElms`, we use `shapes[i % shapes_size]` to choose a shape when i, the loop variable running through number of different elements, is larger than `shapes_size`, the number of different shapes.

In an extreme case, if there is only one shape, for example, a circle, then we will choose different color circles to represent different number of elements.

In another example, supposer there are only two shapes: circle and pentagon, and we need to represent three elements, then we use red circle, green pentagon, yellow circle, where yellow color is represented by `\033[33m`.

```
    std::string shapes[] = {
       "\u2b24", //circle
       "\u2b1f", //pentagon
       "\u25fc", //medium square, can add color
       "\u272f", //star, can add color
       "\u2665", //heart
       "\u25b2", //triangle
    };
    int shapes_size = sizeof(shapes) / sizeof(shapes[0]);

    std::vector<std::string> colorShapeMap(numDiffElms);
    int colorCode = 31;
    //"\033[31m", a const char*, represents red text color
    //"\033[32m" represents green text color
    for (int i = 0; i < numDiffElms; i++) {
        colorShapeMap[i] = "\033[" + std::to_string(colorCode) + "m" + shapes[i %
   shapes_size] + "\033[0m";
        //restore to \033[0m, black color, after displaying a colored shape
        colorCode++;
    }
```

Run the above code when `numDiffElms` is 2. Initialize `colorCode` to be 31.

i	colorShapeMap[i]	i++	colorCode++
0	`"\033[" + std::to_string(colorCode) + "m" + shapes[i % shapes_size] + "\033[0m"` `"\033[31m \u2b24 \033[0m"` (red circle, then restore to black color `\033[0m`)	1	32
1	`"\033[32m \u2b1f \033[0m"` (green pentagon, then restore to black color `\033[0m`)	2	33

Reference: colors are listed in `https://gist.github.com/kamito/704813`. Shapes are taken from

https://en.wikipedia.org/wiki/Geometric_Shapes_(Unicode_block). Note that not every shape can add color, for example, cannot add color to unicode 25fe, a medium square, only black color is available.

6.5 Submit Task B

Based on code of SortGame.cpp in Task A, do the following. Then submit SortGame.cpp to gradescope.

1. In SortGame::SortGame(std::vector<std::vector<int>> binData, int emptyBins), the elements in data member bins are laid out properly already; no need to randomize anymore.

2. Define randomize method. **Call it in constructors** SortGame::SortGame() and

   SortGame::SortGame(int numDiffElms, int capacity, int numEmptyBins).

   Warning: If you do not define code in the default constructor SortGame::SortGame() but call constructor delegate through SortGame::SortGame(int numDiffElms, int capacity, int numEmptyBins), only call randomize once in that non-default constructor.

3. Define display method.

4. Test locally before uploading to gradescope.

   (a) Comment private: line in SortGame.hpp as //private:. This is for debug purpose. Need to uncomment when release the product.

   (b) Comment all occurrences of srand statements in SortGame.cpp.

   (c) Upload SortGame.hpp and SortGame.cpp to https://www.onlinegdb.com/online_c++_compiler. Note that compilers in different operating systems – linux, Mac, windows – may get different random numbers for srand statement. onlinegdb runs in Linux and has the same results in servers of gradescope.

   (d) Edit main.cpp in onlinegdb as follows.

```
#include "SortGame.hpp"
#include <iostream>
#include <string>

void print_bins(std::vector<std::vector<int>> bins);

int main(int argc, const char *argv[]) {
    SortGame *game = nullptr; //elements are not randomized yet

    switch (*argv[1]) {
        case 'A':
        {   //test randomize method
            game = new SortGame(2, 4, 1);
            srand(1); //cannot use other seeds, or with the changing seed in
   constructors of SortGame.cpp, the code will not work.
            game->randomize();

```

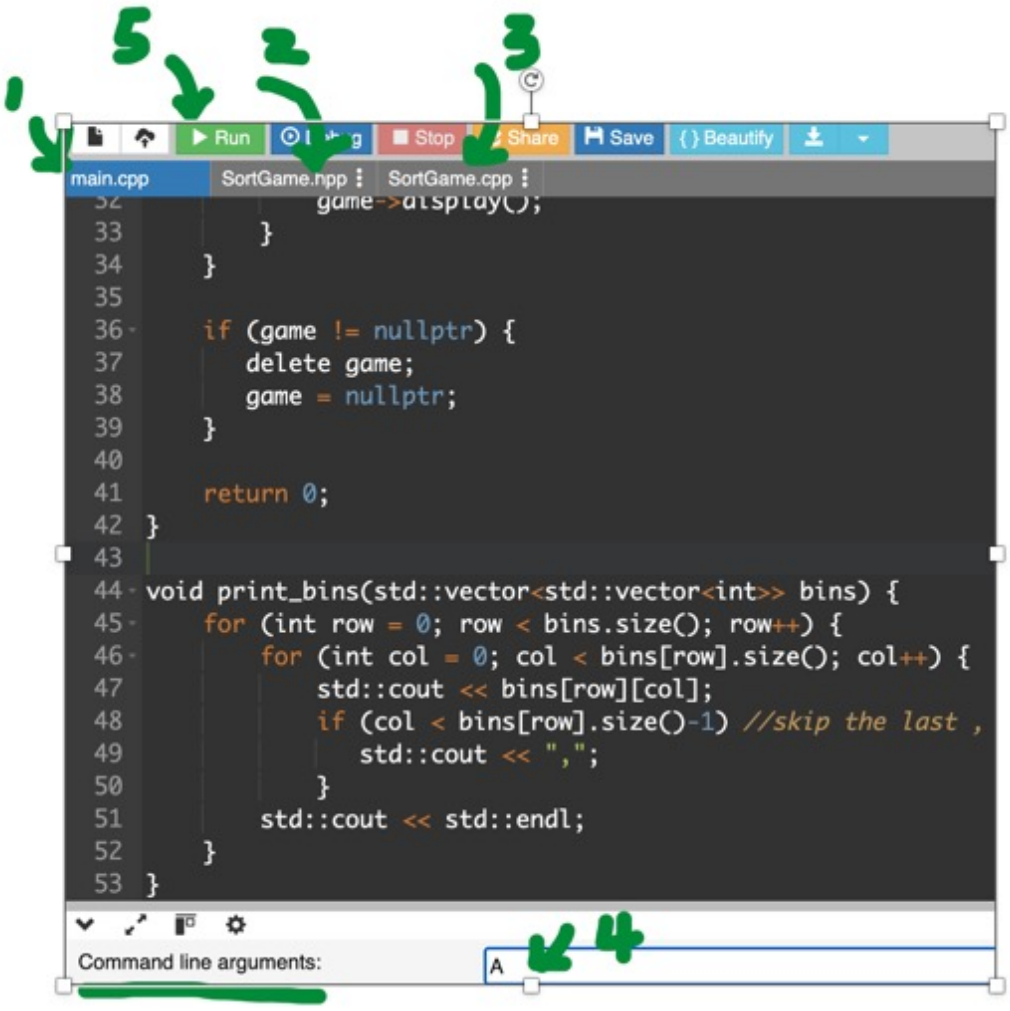
```
17              std::cout << "After randomizing, data member bins is" << std::endl
    ;
18              print_bins(game->bins);
19              break;
20          }
21        case 'B': //display
22          {
23              std::vector<std::vector<int>> binData = {{1, 2, 1, 1}, {2, 2, 1,
    2}};
24              //call SortGame(std::vector<std::vector<int>> binData, int
    numEmptyBins)
25              game = new SortGame(binData, 1);
26
27              //Since the first element of a bin is displayed at the bottom,
28              //while the last element is displayed at the top,
29              //the top element is in the back of bin.
30              //Vector class provides push_back and pop_back operations.
31              //push_back means to add to the top of the bin, a vector object.
32              //pop_back means to remove the top element from a bin, a vector
    object.
33              //Note that the return type of pop_back method is void,
34              //to get the top element of a bin before poppting it out,
35              //need to call back() method first.
36
37              //Use elm to save the top element of game->bins[1], the second bin
38              int elm = game->bins[1].back();
39
40              //pop the top element out from the second bin, denoted by game->
    bins[1]
41              game->bins[1].pop_back();
42              //push elm to the top of the third bin, denoted by game->bins[2]
43              game->bins[2].push_back(elm);
44
45              game->display();
46          }
47      }
48
49      if (game != nullptr) {
50          delete game;
51          game = nullptr;
52      }
53
54      return 0;
55  }
56
57  void print_bins(std::vector<std::vector<int>> bins) {
```

```
    for (int row = 0; row < bins.size(); row++) {
        for (int col = 0; col < bins[row].size(); col++) {
            std::cout << bins[row][col];
            if (col < bins[row].size()-1) //skip the last ,
               std::cout << ",";
            }
        std::cout << std::endl;
    }
}
```

5. Upload SortGame.hpp, with `private:` commented. That is, `//private:`.

6. Upload SortGame.cpp, which defines constructors and methods randomize and display.

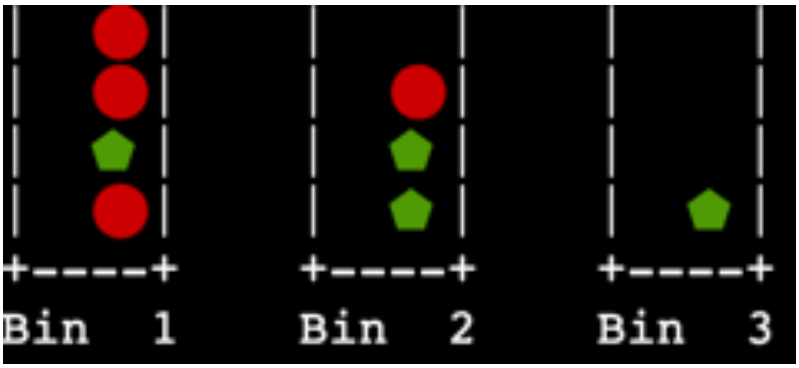
7. When running the code, need to add A or B in Command line arguments. In Steps 1-3, edit `main.cpp` of onlinegdb, upload `SortGame.hpp` and `SortGame.cpp`. In Step 4, put A or B in Command line arguments textbox. In Step 5, click Run button.



8. Put A in Command line arguments, here is a sample output.

```
After randomizing, data member bins is
2,2,1,2
1,1,1,2

```

9. Put B in Command line arguments, here is a sample output.



10. You can also run the code in your local computer by doing the following.

    (a) Generate a runnable file
    g++ -std=c++11 SortGame.cpp main.cpp

    (b) Test randomize method.
    ./a.out A
    However, the output of randomize in your local computer may be different from onlinegdb, which is a linux-based system.
    The following is a sample output from a Mac computer.

```
After randomizing, data member bins is
2,1,1,2
1,2,1,2

```

    (c) Test display method.
    ./a.out B
    Output is the same as onlinegdb, shown in Step 9.

7 Task C: implement move method

Now the shapes are placed randomly in the first `numDiffElms` bins. Hopefully, with the help of `numEmptyBins` many empty bins, we can sort all shapes by placing all elements with the same shape in one bin. However, we do not know which bin may hold which shape in a solution. Also, beware that some puzzles cannot be solved.

The header of `move` is as follows.

```
bool move(std::vector<std::vector<int>>& shapesInBins, int& numFinishedBins)
```

Note that in the original version of `SortGame.hpp`, we use the name `binBalls` in **declaration** of `move` method. In the **defination** of `move` method in `SortGame.cpp` in Task C, we use `shapesInBins` – a more appropriate name – instead of `binBalls`. The change will not affect code running.

In **declaration** of a method, we only care about the types of parameter. In fact, in `SortGame.hpp`, even if we use the following **declaration** of `move` without providing names for formal parameters, the code still runs well. **Warning:** ampersand symbol & after the parameter types cannot be omitted since these parameters are passed by reference.

```
1 bool move(std::vector<std::vector<int>>&, int&);
```

However, in the header of a **definition** of a method, must provide formal parameter names, or we cannot write instructions to work on those variables inside the method.

7.1 Return type and Parameters of `move` method

1. The return type is `bool` type. If input for the move-out bin and the move-in bin are both -1, return false. In this way, we can quit playing the game whenever we like.

2. Parameter `std::vector<std::vector<int>>` records the number of shapes in each bin. The change needs to be carried back the caller of `move` method, so it is pass by refrence – do not forget the & after the parameter type.

   ```
   std::vector<std::vector<int>>& shapesInBins
   ```

   Why do we need `shapesInBins`? Once we move the elements out, the number of occurrences of the top elements in the move out bin is decreased while the number of occurrences of that element in move in bin is increased. By comparing this number with capacity, we can track whether the move in bin is finished sorting or not.

   Without `shapesInBins`, every time some elements are added to the move in bin, we need to run a loop to check whether all the elements in the move in bin are the same or not. We are trade memory – a two dimensional array `shapesInBins` – for efficiency.

3. Parameter `numFinishedBins` records the number of finished bins. A bin is finished if it holds `capacity` many same shape.

   After some moves, number of bins finished can be increased and the change should be carried back to the caller of `move` method. As a result, this parameter is also passed by reference.

   ```
   int& numFinishedBins
   ```

7.2 Intuitive Idea for `move` method

1. After moving the shapes on the top from a bin to the top of another bin, update the number of corresponding shapes in move-out and move-in bins.

2. If one bin is full and contains only the same shape, then this bin is finished. Then increase `numFinishedBins` by 1. Once `numFinishedBins` equals `numDiffElms`, the game finishes successfully.

Warning: declare and initialize `shapesInBins` and `numFinishedBins` are the duties of method `play` – discussed in Task D – the caller of `move` method.

In method `move`, users enter two integers, one represents the label of move-out bin, the other represents the label of the move-in bin. If the move action is valid, update the values of `shapesInBins` and `numFinishedBins` (if necessary), otherwise, give proper prompts.

A move action is valid if

(a) Move-in and move-out bin numbers are valid, that is, both are in [1, `bins.size()`].

(b) The move-out bin is not empty.

(c) One of the following holds.

i. The move-in bin is empty.

ii. The move-in bin is not empty, and

A. the top element in the move-out bin matches to that of the move-out bin.

B. All the same-shape elements on the top of the move-out bin can be moved to the move-in bin without surpassing its capacity.

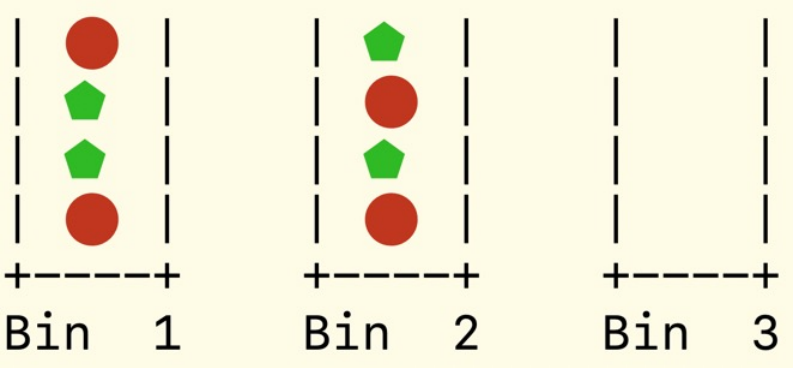
Recall that a function can have at most one return. In method `move`, if a users enters -1 and -1 for both move-out and move-in bin, return false.

However, we need to track the changes of `shapesInBins` and `numFinishedBins` in each move and carry those changes back to the caller – method `play` in our project, so these two parameters are passed by reference.

7.3 An Example of Updating `shapesInBins` and `numFinishedBins` in `move` method

Parameter `shapesInBins` records the number of shapes in each bin, and parameter `numFinishedBins` is the number of bins that are full and hold only the same shape.

1. Suppose we start with the following layout.



In the above example, we have the following values for `shapesInBins` and `numFinishedBins`, declared and initialized in method `play` of Task D.

(a) three bins, indexed by 0, 1, 2.

(b) two shapes, labelled by 1 and 2, respectively, where 1 is interpreted as a red circle, and 2 is interpreted as a green pentagon.

i. For the first bin, there are 2 red circles and two green pentagons. So the vector of integers to represent the number of occurrences of each shape is `{2, 2}`, where the first integer 2 is the number of occurrences of shape labelled by 1 (red circle), and the second integer 2 is the number of occurrences of shape labelled by 2 (green pentagon).

ii. Similarly, for the second bin, the vector of integers to represent the number of occurrences of each shape is `{2, 2}`.

iii. For the third bin – the empty one – the vector of the integers to represent the number of occurrences of each shape is `{0, 0}`.

iv. The following table summaries the above information.

bin index	**# of circle (1st shape)**	**# of pentagon (2nd shape)**	vector
0	2	2	`{2, 2}`
1	2	2	`{2, 2}`
2	0	0	`{0, 0}`

v. The initial value of `numFinishedBins` is 0, since no bin holds only `capacity` many same-shape elements.

2. **Emphasize one more time:** Declare and initialize of `shapesInBins` and `numFinishedBins` in `play` method, the caller of `move` method.

   Method `move` is only responsible for handling a move request and updating pass by reference parameters if necessary.

3. `shapesInBins` is a vector of vector of ints, a two-dimensional array. It is initialized in `play` method.

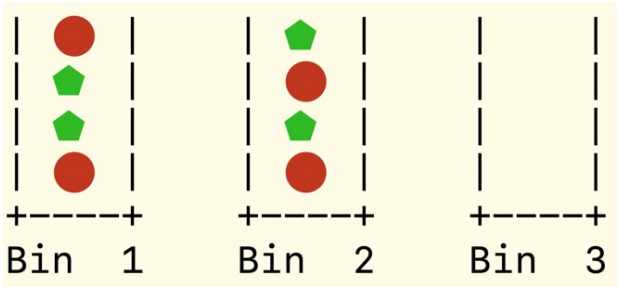
   (a) There are three rows. Each row represents a bin (including the initial empty bin).

   (b) Each row has `numDiffElms` columns.

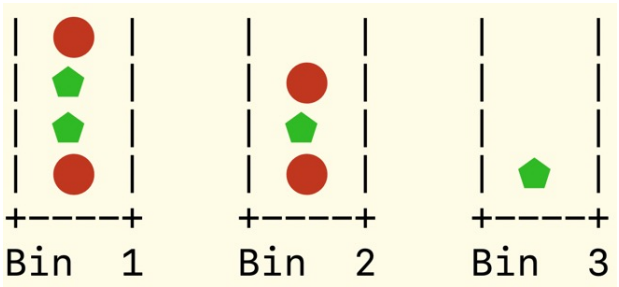
   (c) For the ith row and jth column, where $0 \leq i <$ `bins.size()` and $0 \leq j <$ `numDiffElms`, `shapesInBins`$[i][j]$ is the number of occurrences of jth shape in ith bin.

4. Users enter move out bin as 2 and move in bin as 3 as follows, where black fonts are prompt and highlighted numbers are input.

   Enter the bin to move out and the bin to move in (-1 -1 to stop): 2 3 (with return key)



(a) before moving



(b) after moving

Figure 1: before and after moving from Bin 2 to Bin 3

bin index (aka row index)	# of circle (1st shape) shape index (aka col index) 0	# of pentagon (2nd shape) shape index (aka col index) 1	vector
0	2	2	2, 2
1	2	2 1	2, 1
2	0	1‘	0, 1

5. Update `shapesInBins` as follows.

   (a) Subtract 1 from the move out bin number to get bin index. In our example, we enter 2 for the move out bin. So bin index is 1. That is the row index in two-dimensional array `shapesInBins`.

   (b) Find out the element at the top of the move out bin. The top element of Bin 2 is a pentagon, labelled as shape 2. Subtract 1 from the shape label 2 and get 1. That is the column index in two-dimensional array `shapesInBins`.

   (c) After moving out, `shapesInBins[1][1]` – number of occurrences of Shape 2 in the second bin – is decreased by 1. That is, change from 2 to 1.

   (d) Subtract 1 from the move in bin number to get bin index. Suppose we choose Bin 3 as a move in bin. Then the move in bin index is 3 - 1 = 2.

   (e) The move out bin is empty, so the move action is valid. Increase the number of the corresponding shape – pentagon in our example – by 1. So `shapesInBins[2][1]` is increased from 0 to 1.

7.4 Key ideas of `move` method

In `move` method, enter two integers to represent move out and move in bins, and see whether we can move elements in these two bins.

1. If the entered values are both -1, then the user would like to quit the game, probably realizing that no hope to win.

2. Next, if the entered values are not valid bin numbers, prompt error and return true. Here return true means the users still plan to keep trying and finish the game.

3. Now the inputs for move out and move in bins are valid. We need to check the following scenarios that do not allow move action to finish.

   (a) The move out bin is empty.

   (b) The top element in the move out bin does not match the top element in an non-empty move in bin.

   (c) The move in bin cannot hold all the consecutive same-shape elements in the top of the move out bin.

4. If none of the above situations happen, move the elements from the move out bin to the move in bin.

5. Display the result after the current move finishes.

7.5 Pseudocode of move method

```
bool SortGame::move(std::vector<std::vector<int>>& shapesInBins, int& numFinishedBins) {
    //TODO: Prompt users to enter two integers,
    //the first one is the label of move out bin,
    //the second one is the label of move in bin.
    //A label ranges from 1 to bins.size(),
    //the latter is the number of bins (including empty ones)
    //The prompt must start with Enter or Input.
    //An example of prompt is
    //Enter move out and move in bins (-1 -1 to stop):


    //TODO: declare two integer variables,
    //one for move out bin,
    //the other for move in bin.


    //TODO: enter values for the above two variables.
    //Suppose the variable for label of move out bin is moveOut,
    //the variable for label of move in bin is moveIn.
    //Use cin > moveOut > moveIn;
    //Do not use
    //cin >> moveOut;
    //cin >> moveIn;


    //TODO: if both move out bin and move in bin are -1,
    //return false.



    //TODO: if move out bin or move in bin is not
    //[1, number of bins],
    //where number of bins can be found out by size method of data member bins,
    //print "wrong bin number" (must use the prompt) and return true.


    //Now move out bin and move in bin are valid.
    //find out the corresponding index and put in appropriate variables.
    //TODO: if move out bin is empty, that is, has no element,
    //print out "move out bin is empty". Must use exact the same prompt.
    //Then return true.


```

```
45      //TODO: declare an int variable to hold the top element of move out bin.
46      //Hint: you may use back method from vector class.
47
48
49      //TODO: if the move in bin is not empty and
50      //the top element of the move out bin and
51      //the top element of the move in bin do not match,
52      //print a message that must contain "not match" phrase.
53      //Then return true.
54
55
56
57      //TODO: count the number of consecutive elements
58      //on the top of move out bin.
59      //These are the elements to be moved out.
60      //We use a none-or-all approach,
61      //that is, either all those consecutive elements
62      //on the top of move out bin are sent to move in bin,
63      //or do not move any of those elements at all
64      //if the move in bin cannot hold those many elements.
65
66
67
68      //TODO: if adding the intent-to-move elements from
69      //move out bin will surpass the capacity of move in bin,
70      //print "The move in bin has no sufficient slots.".
71      //Return true.
72
73
74
75      //TODO: move all same-shape elements from the top of move out bin
76      //to the top of move in bin.
77      //Please read Section 7.6 "Hints for move method" for hints.
78      //(1) Update data member bins.
79      //(2) Update parameter shapesInBins.
80      //(2) If after adding the elements, the top element of move in bin
81      //    has capacity many same-shape element,
82      //    increase numFinishedBins by 1.
83      //
84
85
86      //TODO: call display method after moving all elements
87
88      return true;
89  }
```

7.6 Hints for `move` method

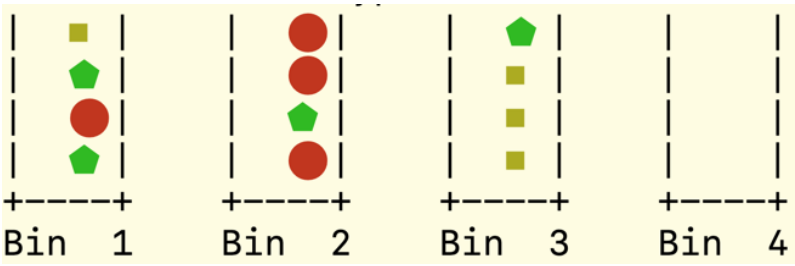
When move happens, we illustrate how to update data member `bins`, and parameters `shapesInBins` and `numFinishedBins` of `move` method.

Before calling `move` method, caller of `move` method needs to define and initialize parameters of `shapesInBins` and `numFinishedBins`, based on data members `bins` and `capacity`.

- `shapesInBins` is a two dimensional array that records the number of shapes in each bin. The first dimension is the index of a bin, the second dimension is the index of a shape.

  Since the labels of bins start from 1, and labelling of shapes also starts from 1, we need to offset by 1 to get the corresponding index in programming.

- `numFinishedBins`, the number of bins with all fully sorted shapes.



shapesInBins — number of shapes in bin

bin number	circle (1)	pentagon (2)	square (3)
1	**1**	**2**	**1**
2	**3**	**1**	**0**
3	**0**	**1**	**3**
4	**0**	**0**	**0**

Explanation:

1. There are four bins and three different shapes in the current game.

   (a) The number of bins is represented by `bins.size()`, where `bins` is a data member representing the distribution of shapes.

   (b) The number of shapes is represented by data member `numDiffElms`.

2. Each bin is represented by a row. There are a total of four bins, so there are four rows. The first row represents the first bin, the second row represents the second bin, and so on.

3. Each column represents a shape. There are three shapes in the current game, so there are three columns. Recall that in `display` method, shape 1 is mapped to a red circle, shape 2 is mapped to a green pentagon, and shape 3 is mapped to a yellow square.

4. In `shapesInBins`, a two dimensional array – shown as a table – the cell at the first row and the second column stores the number of pentagons in the first bin, which is 2.

(a) Since row and column indices start from 0, value `shapesInBins[0][1]` is 2, where the first index is row index 0, representing the first row, and the second index is column index 1, representing the second column.

(b) The first bin (Bin 1) has one circle, two pentagons, and one square. So the first row has values 1, 2, 1.

(c) Similarly, the second bin (Bin 2) has 3 circles, 1 pentagon, and no square. So the second row has values 3, 1, 0.

A bin is finished if it has `capacity` many same-shape elements. In this game, `capacity` is 4. In the beginning of the current game, no bin is finished yet. So argument (aka actual parameter) `numFinished` for `move` method is 0. However, some bins can be sorted in the beginning of other games. As a result, `numFinishedBins` need to be set to an appropriate value.

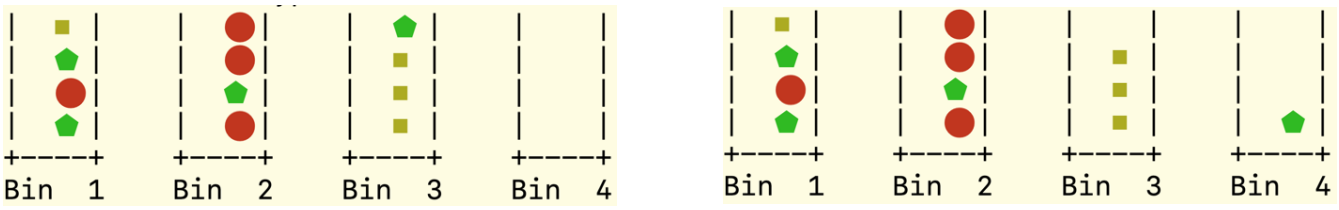
Play the game as follows.

1. Users enter move out bin as 3 and move in bin as 4 as follows, where black fonts are prompt and highlighted numbers are input.

   Enter the bin to move out and the bin to move in (-1 -1 to stop): 3 4 (with return key)

   Note: you can use any prompt as long as Enter or Input is the first word.

   (a) Data member `bins` is updated as follows.



(a) before moving (b) after moving

Figure 2: update of `bins` before and after moving a pentagon from Bin 3 to Bin 4

   (b) `shapesInBins` is updated as follows. After a pentagon is moved out from Bin 3 to Bin 4,

      i. The number of pentagons for Bin 3, residing at the third row (corresponding to Bin 3) and the second column (corresponding to pentagon shape) of `shapesInBins` is decreased by 1, change from 1 to 0.

      ii. The number of pentagons for Bin 4, residing at the fourth row (corresponding to Bin 4) and the second column (corresponding to pentagon shape) of `shapesInBins`, is increased by 1, change from 0 to 1.

   (c) In the move in bin (Bin 4), the number of pentagons is increased to be 1 but it still does not equal `capacity`. Hence, parameter `numDiffElms` is not updated and remains to be 0.

2. Users enter move out bin as 1 and move in bin as 3 as follows, where black fonts are prompt and highlighted numbers are input.

   Enter the bin to move out and the bin to move in (-1 -1 to stop): 1 3 (with return key)

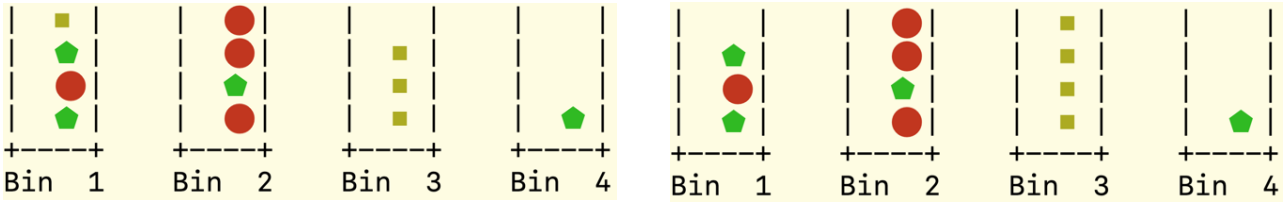
   Note: you can use any prompt as long as Enter or Input is the first word.

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	1	2	1
2	3	1	0
3	0	1	3
4	0	0	0

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	1	2	1
2	3	1	0
3	0	~~1~~ 0	3
4	0	~~0~~ 1	0

Table 1: update of `shapesInBins` after moving a pentagon from Bin 3 to Bin 4

(a) Data member `bins` is updated as follows.



(a) before moving (b) after moving

Figure 3: update of `bins` before and after moving a square from Bin 1 to Bin 3

(b) `shapesInBins` is updated as follows. After a square is moved out from Bin 1 to Bin 3.

i. The number of squares for Bin 1, residing at the first row (corresponding to Bin 1) and the third column (corresponding to square shape) of `shapesInBins` is decreased by 1, change from 1 to 0.

ii. The number of squares for Bin 3, residing at the third row (corresponding to Bin 3) and the third column (corresponding to square shape) of `shapesInBins`, is increased by 1, change from 3 to 4.

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	1	2	1
2	3	1	0
3	0	0	3
4	0	1	0

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	1	2	~~1~~ 0
2	3	1	0
3	0	0	~~3~~ 4
4	0	1	0

Table 2: update of `shapesInBins` after moving a square from Bin 1 to Bin 3

(c) Since the increased number of squares in the move in bin – Bin 3 in this example – equals `capacity`, parameter `numDiffElms` is increased by 1, changing from 0 to 1. We gray all cells in Bin 3 since the bin is finished.

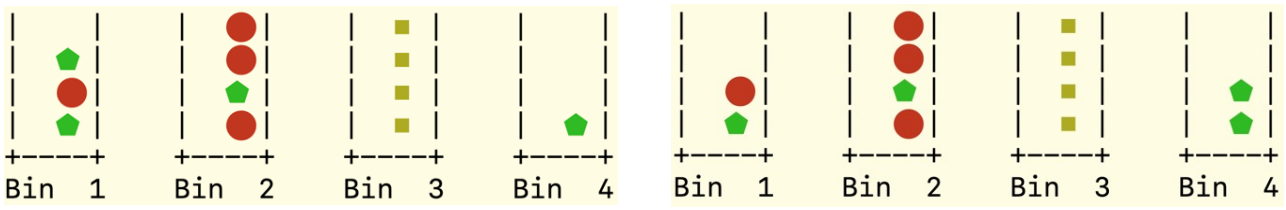
shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	**1**	**2**	**0**
2	**3**	**1**	**0**
3	**0**	**0**	**4**
4	**0**	**1**	**0**

3. Users enter move out bin as 1 and move in bin as 4 as follows, where black fonts are prompt and highlighted numbers are input.

   Enter the bin to move out and the bin to move in (-1 -1 to stop): 1 4 (with return key)

   Note: you can use any prompt as long as Enter or Input is the first word.

   (a) Data member `bins` is updated as follows.



(a) before moving (b) after moving

Figure 4: update of `bins` before and after moving a pentagon from Bin 1 to Bin 4

   (b) `shapesInBins` is updated as follows. After a pentagon is moved out from Bin 1 to Bin 4.

      i. The number of pentagons for Bin 1, residing at the first row (corresponding to Bin 1) and the second column (corresponding to pentagon shape) of `shapesInBins` is decreased by 1, change from 2 to 1.

      ii. The number of pentagons for Bin 4, residing at the fourth row (corresponding to Bin 4) and the second column (corresponding to pentagon shape) of `shapesInBins`, is increased by 1, change from 1 to 2.

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	**1**	**2**	**0**
2	**3**	**1**	**0**
3	**0**	**0**	**4**
4	**0**	**1**	**0**

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	**1**	~~**2**~~ **1**	**0**
2	**3**	**1**	**0**
3	**0**	**0**	**4**
4	**0**	~~**1**~~ **2**	**0**

Table 3: update of `shapesInBins` after moving a pentagon from Bin 1 to Bin 4

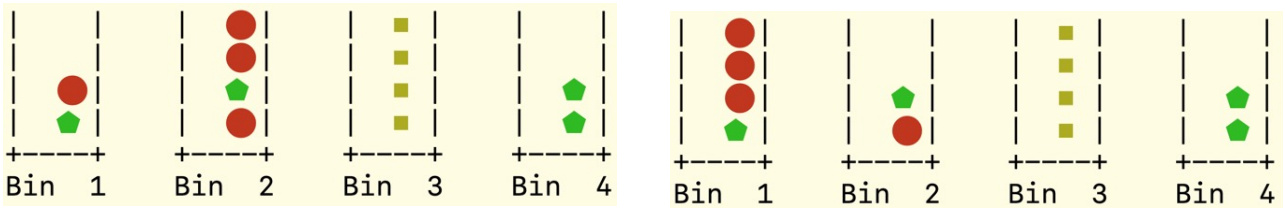
   (c) In the move in bin (Bin 4), the number of pentagons is increased to 2, but it still does not equal `capacity`. Hence, parameter `numDiffElms` is not changed and remains to be 1.

4. Users enter move out bin as 2 and move in bin as 1 as follows, where black fonts are prompt and highlighted numbers are input.

   Enter the bin to move out and the bin to move in (-1 -1 to stop): 2 1 (with return key)

   Note: you can use any prompt as long as Enter or Input is the first word.

   (a) Data member `bins` is updated as follows.



(a) before moving (b) after moving

Figure 5: update of `bins` before and after moving two circles from Bin 2 to Bin 1

   (b) `shapesInBins` is updated as follows. After two circles are moved out from Bin 2 to Bin 1.

      i. The number of circles for Bin 2, residing at the second row (corresponding to Bin 2) and the first column (corresponding to circle shape) of `shapesInBins` is decreased by 2, change from 3 to 1.

      ii. The number of circles for Bin 1, residing at the first row (corresponding to Bin 1) and the first column (corresponding to circle shape) of `shapesInBins`, is increased by 2, change from 0 to 2.

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	**1**	**1**	**0**
2	**3**	**1**	**0**
3	**0**	**0**	**4**
4	**0**	**2**	**0**

shapesInBins	number of shapes in bin		
	circle	pentagon	square
bin number	1	2	3
1	~~1~~ **3**	**1**	**0**
2	~~3~~ **1**	**1**	**0**
3	**0**	**0**	**4**
4	**0**	**2**	**0**

Table 4: update of `shapesInBins` after moving two circles from Bin 2 to Bin 1

   (c) In the move in bin (Bin 1), the number of circles is increased to 3, but it still does not equal `capacity`. Hence, parameter `numDiffElms` is not changed and remains to be 1.

5. Omit the rest moving.

6. In general, these are steps to update `bins`, `shapesInBins`, and `numFinishedBins` in `move` method.

   (a) Make sure that input move out bin and move in bin labels are in [1, `bins.size()`].

   (b) Make sure that the move out bin is not empty.

   (c) Make sure that if the move in bin is not empty, the top element of the move out bin match that of the move in bin.

(d) Make sure that the move in bin can hold all same-shape top elements from the move out bin.

(e) Once the above conditions hold, a move can happen. Update data member `bins`.

i. Find out the move out bin. Get its top element by calling `back()` method on the move out bin. Save the return of `back` method to an integer variable called `top`.

ii. Count the number of consecutive elements with value `top` on the back (aka top) of move out bin. Save the number in variable `count` or whatever name you deem appropriate.

iii. To move out `count` many top elements for the move out bin, call `pop_back()` method for `count` many times for the move out bin.

iv. To move `count` many elements whose value is `top` to the move in bin, call `push_back` method with parameter `top` for `count` many times for the move in bin.

(f) Afterwards, update `shapesInBins`.

i. Decrease the element of `shapesInBins` whose row represents the move out bin and whose column represents the top element by `count`.

ii. Increase the element whose row represents the move in bin and whose column represents the top element by `count`.

iii. Check whether the increased value equals to `capacity` or not. If the answer is yes, then the move in bin has `capacity` many elements with value `top`, increase `numFinishedBins` by 1.

7. **Warning**: in current design, we label bins and shapes from 1. However, the indices of an array starts from 0, so we need to subtract one from bin label or shape value by 1 to get the corresponding indices. This is called **offset by 1 to get index if label starts from 1**.

   Some students suggested to start the labelling of bins and shapes from 0 – instead of 1 – to avoid the need of offset, that is a good idea that can be explored later **after** we update the design of the project.

7.7 Test locally for Task C

Type in the following main.cpp.

```
#include <iostream>
#include <sstream>
#include <vector>
#include <ctime>
#include "SortGame.hpp"
//g++ -std=c++11 SortGame.cpp main.cpp
//test different cases of move method using
//./a.out A
//./a.out B
//...
//./a.out H

void print_bins(std::vector<std::vector<int>> bins, int capacity) {
    for (int row = 0; row < bins.size(); row++) {
       //for (int col = 0; col < bins[row].size(); col++) {
```

```
        for (int col = 0; col < capacity; col++) {
            if (col < bins[row].size())
                std::cout << bins[row][col];
            else std::cout << " ";
            //if (col < bins[row].size()-1) //skip the last ,
            std::cout << ","; //do not skip the last , to see how many items are there
        }
        std::cout << std::endl;
    }
}

std::vector<std::vector<int>> get_shapes_in_bins_numFinishedBins(std::vector<std::vector
    <int>> bins, int capacity, int& numFinishedBins) {
    int numDiffElms = -1; //the maximum label is the number of different elements in the
    bins (may include empty bins)
    for (int i = 0; i < bins.size(); i++)
        for (int j = 0; j < bins[i].size(); j++)
            if (bins[i][j] > numDiffElms)
                numDiffElms = bins[i][j];

    std::vector<std::vector<int>> result;
    for (int i = 0; i < bins.size(); i++) {
        std::vector<int> row(numDiffElms, 0); //put numDiffElms many 0's to row
        result.push_back(row);
    }

    numFinishedBins = 0;
    for (int i = 0; i < bins.size(); i++) {
        for (int j = 0; j < bins[i].size(); j++) {
            result[i][bins[i][j]-1]++;

            if (result[i][bins[i][j]-1] == capacity)
                numFinishedBins++;
        }
    }

    return result;
}
//Warning: if not adding { after case: and } after break;
//Then starting from the third case, we would get an error,
//error: cannot jump from switch statement to this case label.
//Structure your switch statements using curly braces round all of the cases interiors
int main(int argc, const char *argv[]) {
    SortGame *game = nullptr; //elements are not randomized yet

    switch (*argv[1]) {
```

```
60          case 'A':
61          {   //test move method when input -1 -1
62              //print the return,
63              //which should be false
64
65              std::vector<std::vector<int>> binData = {{1, 2, 3}, {3, 2, 1}, {2, 3, 1}};
66              game = new SortGame(binData, 1);
67
68              int numFinishedBins = 0;
69              std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need shapesInBins and numFinishedBins to call move method
70
71              //the following codes inject inputs to cin
72              std::string input = "-1 -1";
73              std::stringstream ss(input);
74
75              std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream
76
77              bool result = game->move(shapesInBins, numFinishedBins);
78              int move_out, move_in;
79              std::cin >> move_out >> move_in;
80              std::cout << "call move method with -1 -1 for move out and move in bins,
    return is" << std::endl;
81              std::cout << std::boolalpha << result << std::endl;
82              break;
83          }
84          case 'B': //bin number is not correct
85          {
86              std::vector<std::vector<int>> binData = {{1, 2, 3}, {3, 2, 1}, {2, 3, 1}};
87              game = new SortGame(binData, 1);
88
89              int numFinishedBins = 0;
90              std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need shapesInBins and numFinishedBins to call move method
91
92              std::string input = "1 " + std::to_string(game->bins.size()+1); //game->
    bins.size() is the maximum bin number we can enter
93              std::stringstream ss(input);
94
95              std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream
96
97              std::cout << "call move method with " << input << " for move out and move
    in bins, output is" << std::endl;
98              bool result = game->move(shapesInBins, numFinishedBins);
```

```
            int move_out, move_in;
            std::cin >> move_out >> move_in;
            break;
        }
        case 'C': //move out bin is empty
        {
            std::vector<std::vector<int>> binData = {{1, 2, 3}, {3, 2, 1}, {2, 3, 1}};
            game = new SortGame(binData, 1);

            int numFinishedBins = 0;
            std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need shapesInBins and numFinishedBins to call move method

            std::string input = "4 1"; //bin 4 is empty
            std::stringstream ss(input);

            std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream

            std::cout << "call move method with " << input << " for move out and move
    in bins, output is" << std::endl;
            bool result = game->move(shapesInBins, numFinishedBins);
            int move_out, move_in;
            std::cin >> move_out >> move_in;
            break;
        }
        case 'D': //top elements do not match
        {
            std::vector<std::vector<int>> binData = {{1, 2, 3}, {3, 2, 1}, {2, 3, 1}};
            game = new SortGame(binData, 1);

            //move top elements from bin 2 (bin index 1) to bin 4 (bin index 3)
            int elm = game->bins[1].back();
            game->bins[1].pop_back();
            game->bins[3].push_back(elm);

            //calculate shapesInBins and numFinished
            int numFinishedBins = 0;
            std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need to call move method
            std::string input = "1 2";
            std::stringstream ss(input);

            std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream

```

```
140             std::cout << "call move method with " << input << " for move out and move
    in bins, output is" << std::endl;
141
142             bool result = game->move(shapesInBins, numFinishedBins);
143             int move_out, move_in;
144             std::cin >> move_out >> move_in;
145             break;
146         }
147         case 'E': //no sufficient slots in move in bin
148         {
149             std::vector<std::vector<int>> binData = {{1, 3, 2}, {1, 2, 1}, {2, 3, 3}};
150             game = new SortGame(binData, 1);
151
152             //move top elements from bin 1 (bin index 0) to bin 4 (bin index 3)
153             int elm = game->bins[0].back();
154             game->bins[0].pop_back();
155             game->bins[3].push_back(elm);
156             //now bins is changed to be {{1, 3}, {1, 2, 1}, {2, 3, 3}, {2}}
157
158             //calculate shapesInBins and numFinished
159             int numFinishedBins = 0;
160             std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need shapesInBins to call move method
161
162             std::string input = "3 1"; //Bin 3 has two 3's but Bin 1 has only one empty
     slot
163             std::stringstream ss(input);
164
165             std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream
166
167             std::cout << "call move method with " << input << " for move out and move
    in bins, output is" << std::endl;
168             bool result = game->move(shapesInBins, numFinishedBins);
169             int move_out, move_in;
170             std::cin >> move_out >> move_in;
171             break;
172         }
173
174         case 'F': //move one element to an empty bin
175          {
176             std::vector<std::vector<int>> binData = {{1, 3, 2}, {1, 2, 1}, {2, 3, 3}};
177             game = new SortGame(binData, 1);
178
179             //calculate shapesInBins and numFinished
180             int numFinishedBins = 0;
```

```
181             std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need to call move method
182
183             std::string input = "2 4"; //Move 1 on top of Bin 2 to top of Bin 4
184             std::stringstream ss(input);
185
186             std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream
187
188             std::cout << "call move method with " << input << " for move out and move
    in bins, output is" << std::endl;
189             bool result = game->move(shapesInBins, numFinishedBins);
190             int move_out, move_in;
191             std::cin >> move_out >> move_in;
192
193             std::cout << "data of bins:" << std::endl;
194             //The following print outs let user get the contents of bins in gradescope
    scripts.
195             //print contents in the first numDiffElms bins
196             print_bins(game->bins, game->capacity);
197
198             std::cout << "data of shapesInBins:" << std::endl;
199             print_bins(shapesInBins, shapesInBins[0].size());
200             std::cout << "number of finished bins:" << std::endl;
201             std::cout << numFinishedBins << std::endl;
202
203             break; //VERY important: give a break in every case
204         }
205         case 'G':
206         {
207             std::vector<std::vector<int>> binData = {{1, 3, 2}, {1, 2, 1}, {2, 3, 3}};
208             game = new SortGame(binData, 1);
209
210             //calculate shapesInBins and numFinished
211             int numFinishedBins = 0;
212             std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need to call move method
213
214             std::string input = "3 4"; //Move 2 elements on top of Bin 3 to top of Bin
    4
215             std::stringstream ss(input);
216
217             std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream
218
219             std::cout << "call move method with " << input << " for move out and move
```

```
    in bins, output is" << std::endl;
220             bool result = game->move(shapesInBins, numFinishedBins);
221             int move_out, move_in;
222             std::cin >> move_out >> move_in;
223 
224             std::cout << "data of bins:" << std::endl;
225             //The following print outs let user get the contents of bins in gradescope
    scripts.
226             //print contents in the first numDiffElms bins
227             print_bins(game->bins, game->capacity);
228 
229             std::cout << "data of shapesInBins:" << std::endl;
230             print_bins(shapesInBins, shapesInBins[0].size());
231             std::cout << "number of finished bins:" << std::endl;
232             std::cout << numFinishedBins << std::endl;
233 
234             break;
235         }
236         case 'H':
237         {
238             std::vector<std::vector<int>> binData = {{1, 2, 3}, {1, 2, 1}, {2, 3, 3}};
239             game = new SortGame(binData, 1);
240 
241             int elm = game->bins[0].back();
242             game->bins[0].pop_back();
243             game->bins[3].push_back(elm);
244             //now the top element of the first bin is moved to the last bin, which was
    originally empty
245 
246             //1,2, ,\n1,2,1,\n2, , ,\n3,3,3,\n
247             //1,1,0,\n2,1,0,\n0,1,0,\n0,0,3,\n
248 
249             //calculate shapesInBins and numFinished
250             int numFinishedBins = 0;
251             std::vector<std::vector<int>> shapesInBins =
    get_shapes_in_bins_numFinishedBins(game->bins, game->capacity, numFinishedBins); //
    need to call move method
252 
253             std::string input = "3 4"; //Move 2 elements on top of Bin 3 to top of Bin
    4
254             std::stringstream ss(input);
255 
256             std::cin.rdbuf(ss.rdbuf()); // Redirect cin to the string stream
257 
258             std::cout << "call move method with " << input << " for move out and move
    in bins, output is" << std::endl;
```

```
            bool result = game->move(shapesInBins, numFinishedBins);
            int move_out, move_in;
            std::cin >> move_out >> move_in;

            std::cout << "data of bins:" << std::endl;
            //The following print outs let user get the contents of bins in gradescope
    scripts.
            //print contents in the first numDiffElms bins
            print_bins(game->bins, game->capacity);

            std::cout << "data of shapesInBins:" << std::endl;
            print_bins(shapesInBins, shapesInBins[0].size());
            std::cout << "number of finished bins:" << std::endl;
            std::cout << numFinishedBins << std::endl;

            break;
        }
    }

    if (game != nullptr) {
        delete game;
        game = nullptr;
    }

    return 0;
}
```

Compile and link the code using
`g++ -std=c++11 SortGame.cpp move.cpp -o move`

7.7.1 Test when input move out bin and move in bin are both -1

Run `./move A`
The expected output is

```
Enter move out and move in bins (-1 -1 to stop): call move method with -1 -1 for move
    out and move in bins, return is
false
```

7.7.2 Test when input move out bin and move in bin are not in [1, bins.size()]

Run `./move B`
The expected output is

```
call move method with 1 5 for move out and move in bins, output is
Enter move out and move in bins (-1 -1 to stop): wrong bin number
```

7.7.3 Test when output bin is empty

Run `./move C`

The expected output is

```
call move method with 4 1 for move out and move in bins, output is
Enter move out and move in bins (-1 -1 to stop): move out bin is empty
```

7.7.4 Test when top element of move out bin and that of move in bin do not match

Run `./move D`

The expected output is

```
call move method with 1 2 for move out and move in bins, output is
Enter move out and move in bins (-1 -1 to stop): the top element in the move out bin
   does not match that of the move in bin.
```

7.7.5 Test when move in bin does not have sufficient slots to hold eligible elements from move out bin

Run `./move E`

The expected output is

```
call move method with 3 1 for move out and move in bins, output is
Enter move out and move in bins (-1 -1 to stop): The move in bin has no sufficient
   slots to hold all 2 same-shape elements in the top of the move out bin
```

7.7.6 Move an element to an empty bin

Run `./move F`

The expected output is

```
call move method with 2 4 for move out and move in bins, output is
Enter move out and move in bins (-1 -1 to stop):
```

```
Bin  1    Bin  2    Bin  3    Bin  4
data of bins:
1,3,2,
1,2, ,
2,3,3,
1, , ,
data of shapesInBins:
1,1,1,
1,1,0,
0,1,2,
1,0,0,
number of finished bins:
0
```

7.7.7 Move two elements

Run `./move G`

The expected output is

```
call move method with 3 4 for move out and move in bins, output is
Enter move out and move in bins (-1 -1 to stop):
|  ⬟ |    |  ● |    |    |    |    |
|  ▪ |    |  ⬟ |    |    |    |  ▪ |
|  ● |    |  ● |    |  ⬟ |    |  ▪ |
+----+    +----+    +----+    +----+
Bin  1    Bin  2    Bin  3    Bin  4
data of bins:
1,3,2,
1,2,1,
2, , ,
3,3, ,
data of shapesInBins:
1,1,1,
2,1,0,
0,1,0,
0,0,2,
number of finished bins:
0
```

7.7.8 Move elements and `numFinishedBins` is increased

Run `./move H`

The expected output is

```
call move method with 3 4 for move out and move in bins, output is
Enter move out and move in bins (-1 -1 to stop):
|    |    |  ● |    |    |    |  ▪ |
|  ⬟ |    |  ⬟ |    |    |    |  ▪ |
|  ● |    |  ● |    |  ⬟ |    |  ▪ |
+----+    +----+    +----+    +----+
Bin  1    Bin  2    Bin  3    Bin  4
data of bins:
1,2, ,
1,2,1,
2, , ,
3,3,3,
data of shapesInBins:
1,1,0,
2,1,0,
0,1,0,
0,0,3,
number of finished bins:
1
```

Once pass all the tests, submit `SortGame.cpp` to gradescope.

8 Task D: define `play` method

8.1 Main steps of `play` method

The main idea to call move method until all shapes are sorted.

1. To call `move` method, we need to initialize `shapesInBins`, a two-dimensional array recording the number of shapes in each bin, and `numFinishedBin`, the number of bins with all sorted shapes.
2. Call `display` method to display the initial layout.
3. Declare and initialize a boolean variable – type `bool` in C++ – `bContinue` to be `true`.
4. Keep on calling `move` method as long as `bContinue` is true and shapes are not fully sorted yet. Hints:
   (a) What is the value of `numFinishedBins` when all the shapes are sorted?
   (b) How to update loop variable `bContinue` inside the loop?
5. Need to track the number of moves.
6. If all shapes are sorted, print out "Congratulations" and the total number of moves to finish sorting.
7. Warning: in `move` method, need to use "Enter" to prompt user to enter move out and move in bin.

8.2 Hints for `play` method

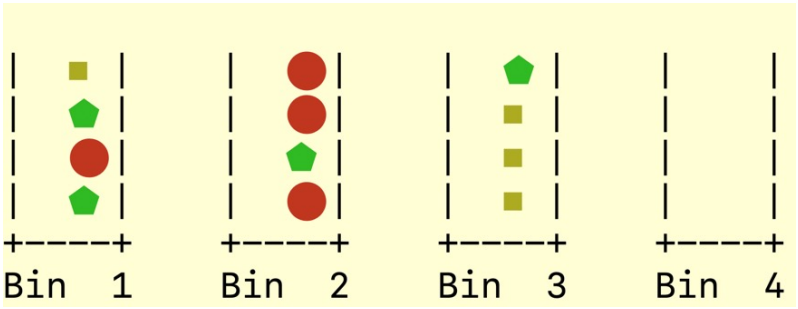
The key is to initialize `shapesInBins` and numFinishedBins in `play` method.

```
void SortGame::play() {
    display();
    //key idea: use 2d array to track the number of same color shapes in each bin
    //once the number of same-color shapes equals capacity,
    //numFinishedBins is increased by 1
    //shapesInBins has bins.size() rows,
    //each row has numDiffElms columns,
    //each column represents a colored shape.

    std::vector<std::vector<int>> shapesInBins;

    //WARNING: the following code results in segmentation errors.
    //std::vector<std::vector<int>> shapesInBins(bins.size());

    //initializ shapesInBins
    for (int i = 0; i < bins.size(); i++) {
        std::vector<int> row(numDiffElms, 0);
        shapesInBins.push_back(row);
    }

    //count the number of same-color shapes in each bin
    //the label of ith bin and jth shape is shapesInBins[i][j],
    //starting from 1, so we need to minus 1.
    int numFinishedBins = 0;
    for (int i = 0; i < bins.size(); i++) {
        for (int j = 0; j < bins[i].size(); j++) { //bins[i][j], and j in [0, bins[i].
   size())
```

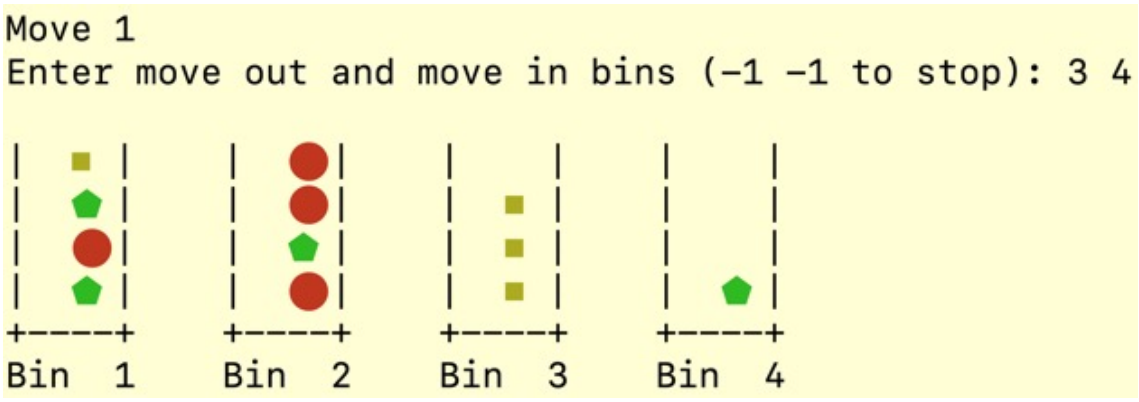
```
            //key:
            //bins[i][j] is a shape in jth column of ith bin,
            //(1) For example, if bins[0][1] is 2,
            //    this means shape 2, which is mapped to a pentagon in display method,
            //    is in the 1st bin -- row index 0 -- and the 2nd column -- column inex
    1 -- of bins.
            //(2) This means the first bin has shape 2,
            //    so the number of shape 2 in the first bin is increased by 1.
            //(3) Note that column index of shapesInBins starts from 0,
            //    so we need to subtract 1 from bins[i][j].
            shapesInBins[i][bins[i][j]-1]++;
            if (shapesInBins[i][bins[i][j]-1] == capacity)
               numFinishedBins++;
        }
    }

    //TODO: initialize numMoves to be an int variable to be zero.


    //TODO: set bContinue to a bool variable with value true.


    //TODO: bContinue to be true and numFinishedBins is smaller than the number of
    shapes
    while (....) {
        //TODO: increase numMoves by 1.


        std::cout << "Move " << numMoves << ":" << std::endl;

        //TODO: put proper parameters to ? and ??
        bContinue = move(?, ??);
    }

    //TODO: fill in condition
    if (???)
       std::cout << "Congratulations! You finish the game in " << numMoves << " moves."
    << std::endl;
}
```

8.3 A sample output

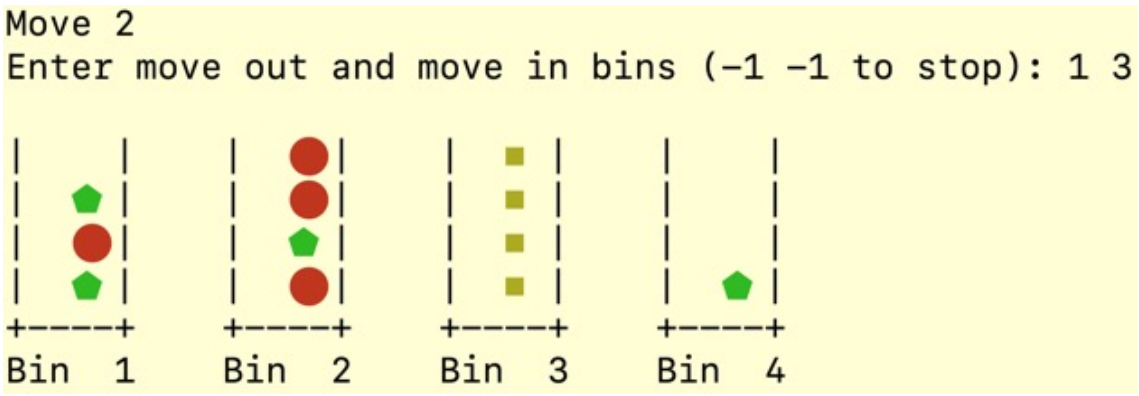
1. Start with the following layout.



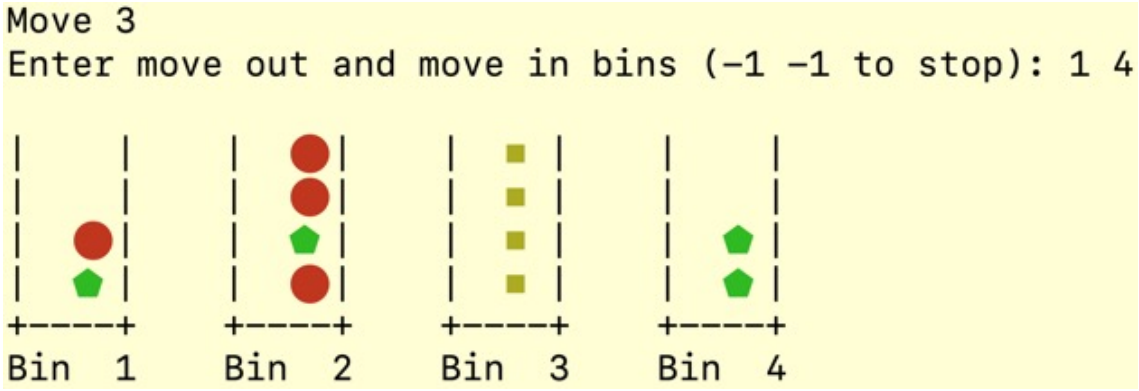
2. First, move from Bin 3 to Bin 4. The result is as follows.



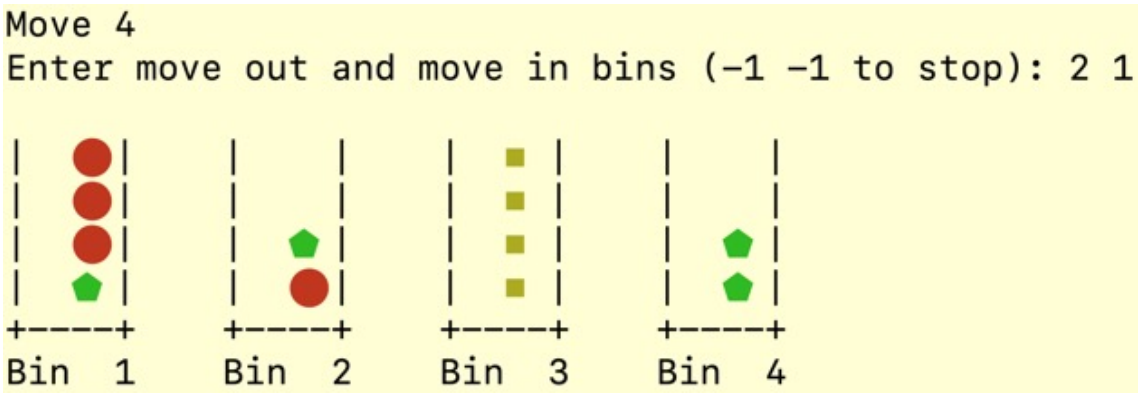
3. Second, move from Bin 1 to Bin 3. The result is as follows.



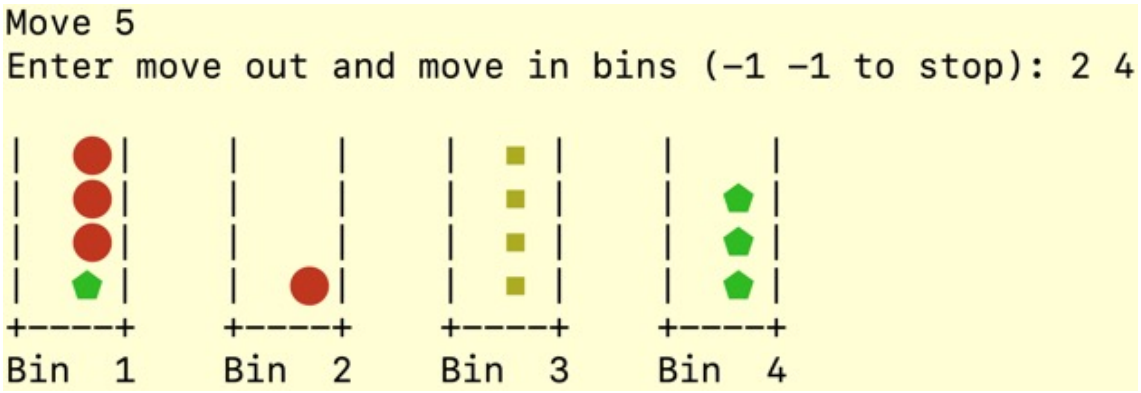
4. Third, move from Bin 1 to Bin 4. The result is as follows.



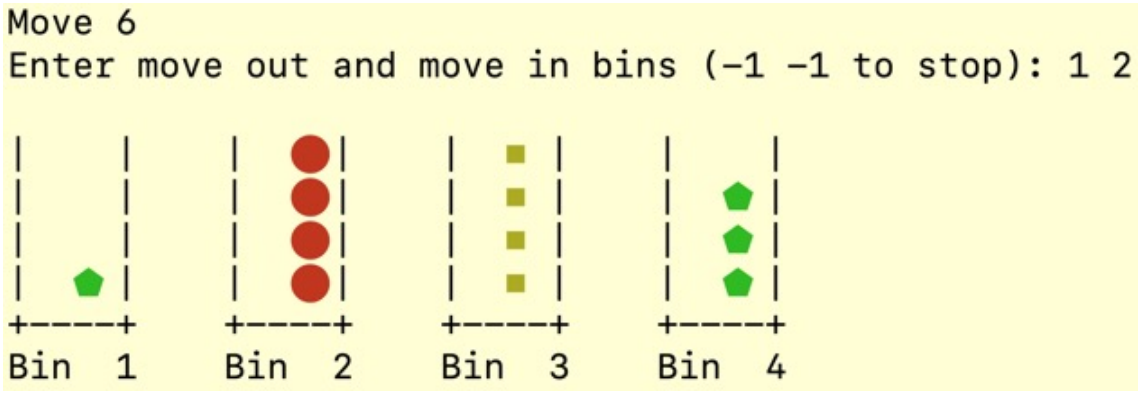
5. Fourth, move from Bin 2 to Bin 1. Note that all elements of the same shape from the move out bin need to be moved to the move in bin. The result is as follows.



6. Fifth, move from Bin 2 to Bin 4. The result is as follows.



7. Sixth, move from Bin 1 to Bin 2. Note that all elements of the same shape from the move out bin need to be moved to the move in bin. The result is as follows.



8. Seventh, move from Bin 1 to Bin 4. Now every shape is sorted, print out "Congratulations!" together with the number of moves taken.

```
Move 7
Enter move out and move in bins (-1 -1 to stop): 1 4

|    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |
+----+    +----+    +----+    +----+
Bin  1    Bin  2    Bin  3    Bin  4
Congratulations! Finish the game in 7 moves.
```

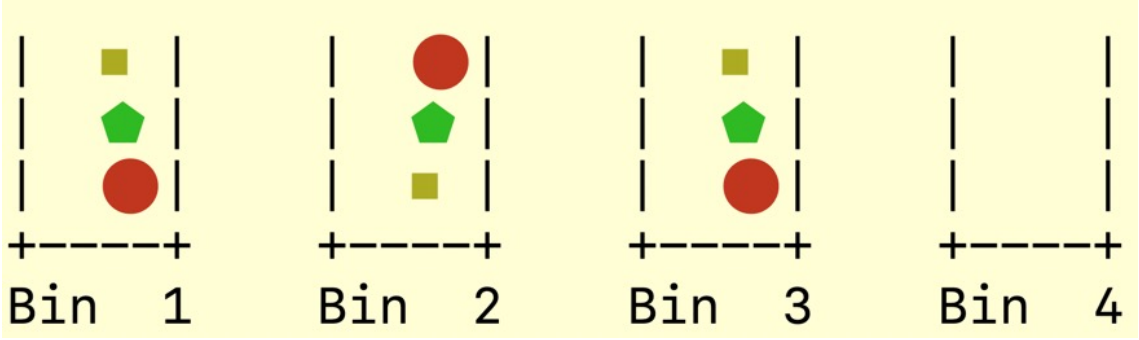
You can test the above example with SortGame.hpp, SortGame.cpp, and the following SortGame-Test.cpp.

```cpp
#include "SortGame.hpp"
#include <iostream>
#include <string>
using namespace std;

int main() {
    vector<vector<int>> binData = { {2, 1, 2, 3}, {1, 2, 1, 1}, {3, 3, 3, 2} };
    //1 for red circle
    //2 for green pentagon
    //3 for yellow square

    //instantiate a SortGame object with the above data and an empty bin
    SortGame game(binData, 1);

    //call play method of game
    game.play();
    return 0;
}
```

To run the code, you can either use make file, shown in Section 10.1, or you can run `g++ -std=c++11 -o sort S` then run `./sort` and do not forget `./` before sort.

9 Not every puzzle can be solved

The following puzzle is unsolvable.



Test the above puzzle using the following SortGameTest.cpp, put in the same folder with SortGame.hpp and SortGame.cpp.

```
#include "SortGame.hpp"
#include <iostream>
#include <string>

int main() {
    std::vector<std::vector<int>> binData = {{1, 2, 3}, {3, 2, 1}, {1, 2, 3}};
    //1 for red circle, 2 for green pentagon, and 3 for yellow square
    SortGame game(binData, 1); //instantiate a game object with binData and an empty
    bin

    //play the game
    game.play();

    return 0;
}
```

Generate a runnable code called `sort` using the following command

```
g++ -std=c++11 -o sort SortGame.cpp SortGameTest.cpp
```

Run the code using

```
./sort
```

10 Wrap up: define SortGameTest.cpp and create makefile

Create `SortGameTest.cpp` with the following contents. The purpose of `SortGameTest.cpp` is to test constructors and methods defined in `SortGame.cpp`.

```
#include "SortGame.hpp"
#include <iostream>
#include <string>
using namespace std;

int main() {
    //TODO: declare a SortGame object called game using its default constructor
```

```
8
9       //TODO: call play method of SortGame object game.
10
11      return 0;
12  }
```

10.1 Use makefile

For a large C++ project, it would better to use makefile, with which, only the modified source codes are recompiled and re-linked.

1. Edit a file called `makefile` with the following contents.

```
1  # This is an example Makefile for sort game project.
2  # This program uses SortGame and SortGameTest modules.
3  # Typing 'make' or 'make run' will create the executable file.
4  #
5
6  # define some Makefile variables for the compiler and compiler flags
7  # to use Makefile variables later in the Makefile: $()
8  #
9  #  -g    adds debugging information to the executable file
10 #  -Wall turns on most, but not all, compiler warnings
11 #
12 # for C++ define  CC = g++
13 CC = g++ -std=c++11
14 #CFLAGS  = -g -Wall
15
16 # typing 'make' will invoke the first target entry in the file
17 # (in this case the default target entry)
18 # you can name this target entry anything, but "default" or "all"
19 # are the most commonly used names by convention
20 #
21 all: run
22
23 # To create the executable file sort (see -o sort), we need the object files
24 # SortGameTest.o and SortGame.o:
25 run:  SortGameTest.o SortGame.o
26     $(CC) -o sort SortGameTest.o SortGame.o
27
28 # To create the object file SortGameTest.o, we need the source
29 # files SortGameTest.cpp, Competition.h
30 SortGameTest.o:  SortGameTest.cpp
31     $(CC) -c SortGameTest.cpp
32
33 # To create the object file SortGame.o, we need the source files
34 # SortGame.cpp.
```

```
35 # By default, $(CC) -c SortGame.cpp generates SortGame.o
36 SortGame.o:  SortGame.cpp
37     $(CC) -c SortGame.cpp
38
39 # To start over from scratch, type 'make clean'.  This
40 # removes the executable file, as well as old .o object
41 # files and *~ backup files:
42 #
43 clean:
44     $(RM) sort *.o *~
```

According to the command in this `makefile`,

`$(CC) -o sort SortGameTest.o SortGame.o`

The generated runnable file is called `sort`, which appears after `-o`.

2. Run make command.

   `make`

3. If there is no error in the above command, run the following command, where dot (.) means current directory.

   `./sort`