# Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of “plagiarism”:

*Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.*

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

**You may…**
• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

**You may not …**
• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one’s account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don’t think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else’s work.

# CS100 Homework 4 (Fall, 2019)

In this homework, you will practice with C++ Object-Oriented Programming (OOP), by re-implementing the dynamic array in homework 2, and making a simple console game.

Percentage of this homework over the whole score: 7%

Submission deadline:

2019-10-23    23:59

**Important: rules on using OJ**

For this homework, there are no specific inputs or outputs, and **main() function is not required** in submission. Therefore, you won't need to worry about output formats. Moreover, you can customize your own main() function and use it for debugging. But note that **you should not submit your main function to OJ, just submit the rest of your codes, otherwise compile errors may happen.**

To encourage debugging locally and not on the OJ output (which is not right for practical programming), the OJ will not be available from the date the homework is released. It will gradually open until the submission deadline passes, and the schedule is:

Part 1 submission will be available on 10/19 at 17:00.
Part 2 submission will be available on 10/21 at 17:00.
You can also see this schedule on OJ homepage.

**Important:**

**Please download hw4package.zip, as it contains skeleton codes and samples for Problem 2, as well as this pdf specification.**

## Problem1. C++ Dynamic Array

Dynamic arrays are a fundamental building block for many practical coding applications. To implement dynamic arrays, you can choose to do it with a C implementation, but it is better to do it with an array class, where you should use dynamic memory allocation with keyword **new**, which has partially been covered in class.

By default, you should always initialize your array with length = 5.

*Important*: You should create this initial dynamic array with keyword **new**, and using static arrays (e.g. int array[5]) in your class will result in a substantial deduction of your score! When your array is full after you add an item (e.g. after adding the 5th item), you should resize your array automatically by doubling its size, in other words,

```
new_size = 2 * original_size
```

As a consequence, the size of your array should be 5, 10, 20, 40, …

Once you need to resize your array, you should output a message indicating how you resized your array in the form given below:

(resize) from 5 to 10

(resize) from 10 to 20

Also, in your destructor, you should use keyword **delete** to destroy the dynamically allocated memory to prevent memory leak. Failing to do so may result in a deduction of your score.

In this part, you are required to implement a dynamic array which contains **integers** using C++. It should perform similarly like C dynamic array you wrote in Homework2. But now, we will implement a "DynamicArray" class to handle the array instead of some global functions.

Your class should have the name of "DynamicArray". It looks like this:

```
class DynamicArray
{
    // Add your member variables and member functions here.
};
```

You need to implement the following member functions:

- `DynamicArray();`

  Initialize your dynamic array and the initial capacity of your array should be 5, and the initial size should be 0.

- `DynamicArray (const DynamicArray & other);`

  The copy constructor. Size and capacity of this array should be equal to the one of "other" , and "data" of the current object should be **a copy** of "other".

  **Note: if you have an array A, and you construct B using this copy constructor (i.e. you write something like "`DynamicArray B(A);`"), then if you change some data in A, the data in B should not be changed.**

- `~DynamicArray ();`

  The destructor: **free all memory** you allocated.

- `void assign (const DynamicArray & other);`

  Assign size of the current object to the size of the "other", and copy "other" 's data to this array, very similar as the copy constructor.

  For example, You may have two dynamic arrays A and B, you can call "`A.assign(B)`" to make A be a copy of B.

  Note: If you then change some data in A, the data in B should not be changed.

- `int getSize() const;`

  Returns current size of this array

  .

- `int getCapacity() const;`

  Returns current capacity of this array.

- `int at (int index) const;`
  Returns the value in the array at position “index”. Its function is similar to accessing an element in an array with [] (e.g. A[0]).
  For example, an array with size 4 is like this: A = [19, 26, 8, 17]; Then, “`A.at(0)`” will return 19, and “`A.at(2)`” will return 8.

- `void push (const int item);`
  Push the given “item” at the end of the array. After pushing the item, you should check whether your array is full. If so, resize its capacity to **`2*origin_capacity`**. Rules about resizing is the same as Homework2, and you can refer to it at the beginning of this part.

- `bool remove (const int index);`
  Remove the item at the position “index”, if the given index is valid, remove the item at “index” and return true; Otherwise (e.g. index < 0), do nothing and return false.
  To make your array continues, if you remove an interiors element, you should shift forward all elements behind it.
  For example,
  A = [0 ,2, 4, 6, 8, 10, 12] is like this:
  ```
  -------------------------------
  | 0 | 2 | 4 | 6 | 8 | 10 | 12 |
  -------------------------------
  ```
  If you call “`A.remove(2);`”, you should remove “4” for A[2] = 4, and the array may be like this:
  ```
  ---------------------------
  | 0 | 2 | 6 | 8 | 10 | 12 |
  ---------------------------
  ```
  Note that “6”, “8”, “10” and “12” are moved forward.

  **Note: In this function, you are not required to resize your array. If an array with size = 6 and capacity = 10, remove 2 elements in it will make its size = 4 and capacity is still 10.**

**<u>What you need to do:</u>**

In this part, you will implement a DynamicArray class using C++. You need to implement the following functions above and make sure that the “`data`”, “`size`” and “`capacity`” can only be accessed inside the function “`at`”, “`getSize`” and “`getCapacity`” functions (i.e. these variables should not be accessed externally by class object).
**Hint:** To satisfy this property, you should recall the keyword “**public**” and “**private**” which has been covered in class.

**Submitting details:**

When submitting this part, you only need to submit the DynamicArray class and all functions you implemented, DO NOT submit “main()” function. Otherwise, compilation errors will occur.

## Problem2. Lunch Delivery

### *Overview*

Every day at noon, hungry Shanghaitech students would swarm into FamilyMart to buy box meals for lunch. In real life, they would get the meal by themselves, but in this story, a miserable employee at Fmart delivers meals to each student, and unfortunately this person is you.

What you are going to write for this problem is a game based on the scenario above. In this game, you play from the perspective of an employee, trying to deliver correct box meals to students. **The sample version is on piazza, along with this pdf. We recommend you to download and play the game to get an idea of what it looks like.** The game interface looks like this:

```
.......S..
P........S
.....S....
```

### *Game Rules:*

The 3*10 grid represents Fmart, where hungry students enter from the right side and walk towards the left side. The positions of students are marked by each 'S', and 'P' represents you. Your goal is to deliver correct meals to students, and to prevent students from reaching the left-most column (the column you are at).

There are three kinds of boxed meals at Fmart: Apple pie, Beef noodles, and Curry rice. They are denoted by A, B, and C in short. Each student would like to have only one kind of meal. For some reasons, the students will not tell you which kind of meal they want, but will only react to your actions. You would pick up a boxed meal, and ask if anyone wants to have it. If a student is given correct meal, he/she will leave Fmart happily.

Your player has 6 different options to make a move. You can choose by entering "a", "b", "c", "u", "d", or "g".

"a", "b", and "c" each means that you pick up a box of Apple pie, Beef noodles, or Curry rice. You also ask if anyone wants the box meal you picked. If someone wants it, their display will change from 'S' to '@'

"u" or "d" means that you move upwards or downwards by a row.

"g" means you give the meal you are holding to the first student in front of you. If that student wants the meal, he/she will leave Fmart, and you can receive 10 points in the game. If that student does not want the meal, or you are not holding any meal, nothing happens.

After you have made a move, all students will move left by one step. If a student has reached the left-most column, he/she will leave Fmart angrily. As a result, you are scolded by other Fmart employees, and will "lose a life" in the game. If you lose 3 lives, you will be fired, and the game is over.

## *Code Implementation:*

This is an Object-Oriented game. (Almost) Everything in the game is considered an object, which has member variables and can perform actions through member functions.

You need to implement classes for Player, Student, and Fmart. They each have own member variables and functions. We have provided a skeleton for you to understand what they are, and you should finish the rest of the code to get it all well going.

To make this specification clear, detailed requirement of each function is not written here, but in the comments of the corresponding functions. Please read the comments carefully.

Here is the basic logic of this game:

The grid is labeled by rows 1 to 3, columns(cols) 1 to 10. (It does not start from 0!)
When the game initializes, the Player ‘P’ is at row 2, col 1. A student is generated at col 10 but a random row, and the type of meal he/she wants is also random.
The game is played in turns. Turns are represented by a loop in Fmart::playGame().

At the beginning of each turn, Fmart::display() is called. This function displays the Fmart and everyone’s position and status. Here’s how everything should be displayed:

> Empty positions should be marked by a dot ‘.’.
> If you are not holding a box meal, then you are marked by ‘P’ in display, and students are marked by ‘S’.
> If you are holding a box meal, then you are marked by ‘A’, ‘B’, or ‘C’, based on the type of meal you are holding, and the students who want this type of meal are marked by ‘@’.

(Do not worry too much, as we have provided much of the implementation of display(), with only a few blanks for you to fill in.)

Then, you can enter a choice in “a”, “b”, “c”, “u”, “d”, and “g” to make a move.

After you have performed your move, Student::moveForward() is called for each student to make a move. Each student will move one step left. If a student reaches col 1, you will lose one life, and this student should be deleted from the game.

After all students have made their moves, there is a 30% probability that a new student is generated. Also, the type of meal that he/she wants should also be random. You should use the randInt(int min, int max) function provided, which returns a random integer in [min, max], inclusively.

## *What you need to do:*

Basically, you need to complete all parts marked by “TODO:” in the skeleton code. You need to make sure that when your code is running, your game’s behavior is the same as the sample we provided, and well moves would not result in errors.

If you feel that provided member variables and functions are not enough, feel free to add your own variables or functions, as long as you do not change the name, parameter list, and return types of everything we provided.

If you have anything uncertain, you can refer to the sample game to check if your game behaves the same as the sample.

## *When Submitting:*

Please submit all except your “main()” function. This includes your #include lines, #define lines, and all classes and functions you wrote. DO NOT submit “main()” function. Otherwise, compilation errors may occur.