VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMMING FUNDAMENTALS - CO1027

ASSIGNMENT 2

THE SWORD IN THE STONE

Version 1.0

Ho Chi Minh City, 03/2023

ASSIGNMENT'S SPECIFICATION

Version 1.1

1 Assignment's outcome

After completing this assignment, students review and make good use of:

- Pointer
- Object Oriented Programming (OOP)
- Singly Linked List.

2 Introduction

Ultimecia, the most powerful witch of all time, due to suffering in love, has decided to destroy all of humanity with time-compression magic, thereby merging past, present, and future. Without the past, present, and future, there will be no nostalgia, ambition, and development, and humanity will cease to exist.

To execute the time compression spell, Ultimecia planted a magic rose. The petals of the rose will gradually fall over time. When the last petal falls, the time compression spell will be complete. To ensure the safety of the rose, Ultimecia hid it behind a dark maze with many monsters to prevent any Knights from approaching. If any Knight can overcome the dark maze and defeat Ultimecia at the end of the labyrinth, the spell will be canceled, and humanity will be saved from the apocalypse.

To defeat Ultimecia, one must possess King Arthur's Excalibur sword. After defeating Bowser and rescuing Princess Guinevere, King Arthur plunged the Excalibur sword into a stone; retiring from his throne and life as a knight, he and Princess Guinevere lived in seclusion in the wilderness. To retrieve the Excalibur sword from the stone, one must have three artifacts: the Paladin's shield, Lancelot's spear, and a strand of Guinevere's hair. These artifacts are hidden in different places within the dark maze.

Even if the sword is retrieved from the stone, only a Knights of the Round Table can use Excalibur. Moreover, a lone knight cannot defeat countless monsters and Ultimecia's powerful magic. Therefore, a group of talented Knights has gathered to undertake a new mission full of challenges and glory: to retrieve the sword from the stone, defeat Ultimecia, and save humanity.

Will the Knights be able to retrieve the sword from the stone? Will Ultimecia abandon her extreme intentions in her suffering? All will be answered in this assignment.

3 Input data

The program's input is contained in two input files for the Army of Knights and the events in the journey to defeat Ultimecia.

3.1 Army of Knights

The data file name for the Army of Knights will be stored in the variable **file_armyknights**. This file has the following format:

- Line 1 contains a positive integer n, $n <= 1000$, representing the number of Knights of the Round Table in the Army of Knights fighting Ultimecia.
- On n next lines, each line is a string describing information for a Knight of the Round Table. The string has the following format:

  ```
  HP␣level␣phoenixdownI␣gil␣antidote
  ```

  In which:

  - ***HP***: The knight's health stat, which is an integer between 1 and 999. This value is also the knight's maximum health ***MaxHP***.
  - ***level***: Knight's level, an integer between 1 and 10.
  - ***phoenixdownI***: The number of type 1 PhoenixDown tears a knight carries, an integer between 0 and 5.
  - ***gil***: The amount of money the knight carries, an integer between 1 and 999.
  - ***antidote***: The number of antidotes carried by the knight, an integer between 0 and 5.

3.2 Events

The input file name for the events will be stored in the variable **file_events**. This file has the following format:

- Line 1 contains a positive integer e, $e <= 1000$, representing the number of events the knight army will encounter.
- The next line contains e events; each event is indexed starting from 1. A numeric value will describe each event, called the event code. The respective meaning of each event is described in Section **??**. An event can happen more than once. Ultimecia will front and protect the rose from the army of Knights. Therefore, event code 99 meets Ultimecia occurs only once and is the last event. Events 95, 96, 97, 98 are guaranteed to happen at least once.

4 Description

The journey to fight Ultimecia goes through many events. The respective meaning of each event is described in Table 1. Depending on the events taking place along the way, the information of the Army of Knights will change. The way the Army fights is that the Knight at the end will fight his opponent, see also Section 5.3. If an item is encountered, the Knight at the end of the Army will try to put the item in his inventory. If the item cannot be placed in the inventory, that Knight will give the item to the Knight in front and the Knight in the front will continue to try to put it in his inventory. This process repeats until the first Knight of the Army. If the first Knight of the Army cannot also put it in the inventory, the item will be discarded. On the other hand, if the last knight receives more than 999 gil, the excess gil will also be passed back to the knight in front, like passing back items.

Information of events is as follows:

1. **If an event with codes 1 to 5** is encountered, the knight must engage the corresponding opponent. If the knight's ***level*** is greater than or equal to the opponent's ***levelO***, the knight wins. Each time he defeats an opponent, the knight receives a corresponding amount, described in 2, however the knight's gil never exceeds 999.
   If the knight's level is less than the opponent's levelO, the knight's HP will be recalculated as follows:

$$HP = HP - baseDamage * (levelO - level) \tag{1}$$

   In which, baseDamage will depend on the opponent, as described in Table 3, the $levelO$ of the Opponent is calculated as follows:

$$levelO = (i + event_{id})\%10 + 1$$

Table 1: Events on the journey to Ultimecia

Event Code	Meaning
1	Meet MadBear
2	Meet Bandit
3	Meet LordLupin
4	Meet Elf
5	Meet Troll
6	Meet Tornbery the Ghost
7	Meet Queen of Cards
8	Meet Nina de Rings the Funny Merchant
9	Meet Durian Garden
10	Meet Omega Weapon the Monster
11	Meet Hades
112	Picked up PhoenixDown type II
113	Picked up PhoenixDown type III
114	Picked up PhoenixDown type IV
95	Picked up Paladin's Shield
96	Picked up Lancelot's Spear
97	Picked up Guinevere's Hair
98	Meet Excalibur Sword
99	Meet Ultimecia

Where, i is the order of events from 0, $event_{id}$ is the event code.
If HP is less than or equal to 0 after calculating with Formula 1, the program needs to perform the following steps:

- Step 1: The knight searches for PhoenixDown in the bag in order from the first position to the last, with the first position being the most recently received item. If the knight found the PhoenixDown, he uses it to restore the HP. But, if the bag is searched completely and the HP is still less than or equal to 0, proceed to Step 2.
- Step 2: If the knight's gil is currently greater than or equal to 100, the knight will call upon the Phoenix to revive himself. When calling upon the Phoenix, the knight's gil will be reduced by 100. The knight's HP will be half his MaxHP (integer division only) upon revival. If the knight cannot be revived due to having less than 100 gil, the knight will be fallen.

2. **If Tornbery is encountered (event code 6)**, the knight will engage Tornbery. Combat is the same as described in item 1. If the knight wins, the knight's level will increase by 1 unit but cannot increase by more than 10. If the knight loses, the knight will be poisoned. As soon as he is poisoned, if the knight has antidote (antidote ¿= 1), the knight will automatically take this medicine and return to normal; the knight's antidote

Table 2: gil obtained when defeating an opponent

Opponent	gil
MadBear	100
Bandit	150
LordLupin	450
Elf	750
Troll	800

Table 3: baseDamage stats of opponents

Opponent	baseDamage
MadBear	10
Bandit	15
LordLupin	45
Elf	75
Troll	95

is reduced by 1. If there is no antidote, Knights will be poisoned as follows:

- Knight drops from his inventory the latest 3 items. If the existing inventory is less than or equal to 3 items, the inventory will become empty.
- HP is reduced by 10; if HP becomes less than or equal to 0, do as described in the item 1.

3. **If encountering Queen of Cards**, the knight will engage the queen. The fighting style is similar to the one described in the first section. If the knight wins, the knight's gil will be doubled. If the gil exceeds 999, pass the excess gil to the front knight. This process repeats to the first knight. If the first knight can not keep more, the excess gil will be discarded. If the knight loses, his gil will be halved (integers only).
4. **If encountering Nina de Rings**, the knight carries out the trades in the order described as follows:

   - Knight will continue his journey and will not trade at all if he has less than 50 gil.
   - If the knight's HP is less than 1/3 of MaxHP (integers only), the knight will give Nina 50 gil which will increase the knight's HP by an amount equal to 1/5 of MaxHP.

5. **If the Army of Knights gets lost in the Durian Garden**, the knight's HP will be restored to MaxHP.
6. **Knights may encounter Omega Weapon** , a prehistoric monster that has existed since the beginning of the universe. If encountered with Omega Weapon, the knight will be defeated at any level, the knight's HP will be reduced to 0 and must take the steps

described in the event 1. Only a knight at level 10 and HP being MaxHP, or Dragon Knight at any level can defeat Omega Weapon. In case of victory, the knight's level will be increased to 10, the knight's gil will be increased to 999. Once defeated, the Omega Weapon will never appear again. If the knight encounters the Omega Weapon event again, the knight will skip the event and move on.

7. **If the Army of Knights meets Hades**, the knight will be defeated at any level. Only a knight at level 10 or a Paladin Knight of level 8 and above can defeat Hades. If he defeats Hades, Hades will forge Paladin's shield for the knight. After being defeated, Hades will not appear again. If the knight encounters the event with Hades, the knight will ignore this event and continue.
8. **If the Army of Knights meets Ultimecia**, the fight will take place as follows. If the Knights Army had the Excalibur Sword, the Knights would defeat Ultimecia.
   Without the Excalibur Sword, but three treasures (Shield, Spear, and Hair) are obtained, Ultimecia will agree to fight the Dragon Knights, Paladins, and Lancelots in the Army. Each knight of one of those three types in the army will take turns against Ultimecia in order from the last of the Army to the top one. The fighting method is that each Knight deals a certain amount of damage to Ultmecia's HP. If Ultimecia's HP is zero before the Army of Knights runs out of knights that can fight Ultimecia, the Army wins, otherwise, the Army loses. Ultimecia's HP is initially **5000**. Each type of Knight deals the following amount of damage:

$$damage = HP * level * knightBaseDamage$$

   In which, damage is only taken as an integer if the result is a real number, knightBaseDamage is calculated as Table 4. After dealing damage to Ultimecia, if Ultimecia has not been defeated, the knight will be attacked and be fallen by Ultimecia spells. Then, the suitable knight in the back will continue to take over and attack Ultimecia
   If the Army does not have all 3 treasures, the Army will lose to Ultimecia. When the army loses, all the Knights will die making the number of knights of Army equal to 0.

Table 4: KnightBaseDamage stats of different types of knights

Knight Type	knightBaseDamage
Lancelot	0.05
Paladin	0.06
Dragon Knight	0.075

9. **If the Army of Knights encounters phoenix tears (event code 112, 113, 114**,

these phoenix tears are from previous Knights who fell. When fighting Ultimecia, due to being in the dark labyrinth for a long time, the effect of phoenix tears is no longer the same as before but degrades into 3 different types: II, III, IV. Here are the effects of each types of phoenix tears, including type I, which is the kind that the Knight carries before embarking on his journey:

- Type I: can only be used when ***HP*** <= 0, restore ***HP*** to ***MaxHP***.
- Type II: can be used when ***HP*** < ***MaxHP***/4 (integers only), restoring ***HP*** to ***MaxHP***.
- Type III: can be used when ***HP*** < ***MaxHP***/3 (integer parts only), if ***HP*** <= 0, restore ***HP*** to ***MaxHP***/3 (only take integer part), otherwise increase by ***MaxHP***/4 (take only the integer part).
- Type IV: can be used when ***HP*** < ***MaxHP***/2 (integer parts only), if ***HP*** <= 0, restore ***HP*** to ***MaxHP***/2 (only take integer part), otherwise increase by ***MaxHP***/5 (take only the integer part).

During events, after the fight and ***HP*** are reduced, the knight will, in turn, search his bag for the first usable phoenix tear and use it immediately. The search method is to go from the first item (corresponding to the beginning of the list) to the last item (corresponding to the end of the list), see how to implement the bag using a singly linked list in Item 5.6.

10. **If the Amry of Knights encounters treasures (event codes 95, 96, 97)**, the Army will pick up these treasures.
11. **When the Army of Knights encounters the Excalibur Sword (event code 98)**, if the three treasures are not present, the knight cannot draw the sword and will continue the journey to the next event. If all three treasures are collected, the knight will draw the Excalibur Sword and continue his journey to Ultimecia.
12. **If the battling knight is Lancelot**, the knight will defeat opponents with event codes 1 to 5.
13. **If the battling knight is a Dragon Knight**, due to Dragon's Blood, the knight is not poisoned when he loses to Tornbery.
14. **If the battling knight is a Paladin**, the knight will defeat opponents with event codes 1 to 5. If losing to the Queen of Cards, Paladins do not lose their gil. Nina will also agree to trade even if the Paladin has less than 50 gil and will not collect 50 gil for helping the Paladin increase HP.

5 OOP Classes

This assignment uses Object Oriented Programming to describe the Journey of Fighting Ultimecia. Objects in Object Oriented Programming are represented by class and are described as below.

5.1 List of events

The class **Events** represents the events of the journey against Ultimecia. This class has the following methods:

- Constructor has one parameter type string containing a filename describing information for the events, see Section 3.2 for the file format. This constructor allocates an array of integers to store the event code. Declaration of constructor:

```
Events(const string & file_events);
```

- Method **count** returns the number of events. Method declaration:

```
int count() const;
```

- Method **get** returns the event code at position i (position from 0).

```
int get(int i) const;
```

- The destructor needs to free dynamically allocated memory in heap memory.

5.2 Knight

In the Army of Knights going to fight Ultimecia, there are 4 types of Knights:

1. Paladin: Knight whose initial HP is a prime number.
2. Lancelot: Knight has an initial HP of 888.
3. Dragon Knight: The original HP Knight has exactly 3 digits and these 3 digits form the Pythagorean triple. A Pythagorean triple is a set of 3 positive integers a, b, c such that $a^2 + b^2 = c^2$
4. Regular Knight of the Round Table: None of the above knight types.

Requirement: Students should implement the following classes to represent information for the Knights.

1. class **BaseKnight**: abstract class representing information for a knight. All properties in this class have the access modifier of protected, which:
   - id: an identifier for knight, type int.
   - hp: knight's ***HP*** stat, type int.
   - maxhp: knight's ***MaxHP*** stat, type int.
   - level: ***level*** of Knight, type int.
   - gil: the current amount of the Knight, type int.
   - bag: represents the Knight's item bag, type BaseBag, described later in Section 5.6.
   - knightType: represents the type of Knight, the enum type KnightType includes the values: PALADIN, LANCELOT, DRAGON, NORMAL corresponding to 4 types of Knights: Paladin, Lancelot, Dragon Knight and Ordinary Knight.

   class **BaseKnight** has a static method **create** that takes the Knight's stats and creates the appropriate Knight object. The declaration of this method is as follows:

```
static BaseKnight * create(int id, int maxhp, int level, int gil, int
    antidote, int phoenixdownI);
```

2. class **PaladinKnight**, class **LancelotKnight**, class **DragonKnight**, class **NormalKnight** represent Paladin, Lancelot, Dragon Knight and Regular Knight of the Round Table, respectively. These 4 classes all inherit from the BaseKnight class and must redefine the **fight** method accordingly.

5.3 Army of Knights

The class **ArmyKnights** represents information for an Army of many Knights. These Knights are stored using a dynamically allocated array. This class has the following methods:

- The constructor has one parameter type string containing a file name describing information for the Army, see Section 3.1 for the input data of this file. This constructor initializes an array of Knights from the file's data read in. The id of the first Knight read in is 1, the id of each subsequent Knight equals the previous Knight's id plus 1. The first Knight read-in is at the top of the array. The last knight is read at the last position of the array. The Knight in the last place will be the Knight that fights with the opponents and also the Knight who picks up the item. When this Knight falls, the adjacent Knight standing in front of this Knight will be the next Knight to pick up items and fight. This process continues until the array has no more Knights. Declaration of the constructor:

```
ArmyKnights(const string & file_armyknights);
```

- The destructor needs to free dynamically allocated memory in the heap memory.
- Method *fight* describes how a Knight engages an opponent and performs updates to the Knight. The method returns **false** if the Knight falls in combat, otherwise returns **true**:

```
virtual bool fight(BaseOpponent * opponent);
```

- Method **adventure** accepts a pointer to the Events object, which is a class that holds the events in the journey. This method describes how the Army goes through each event. At the end of each event, the Army information needs to be printed using the printInfo() method of the ArmyKnights class. The method returns **true** if the Knights Army defeats Ultimecia; otherwise, the method returns **false**. The method declaration is as follows:

```
bool adventure(Events * events);
```

- Method **count** returns the current number of Knights in the Army of Knights. Note that if a Knight falls, the number of Knights will be reduced by 1. The declaration of the method is as follows:

```
int count() const;
```

- Method **lastKnight** returns a pointer to the last Knight object in the Knight Army. If the Army has no Knights, return **NULL**. The method declaration is as follows:

```
BaseKnight * lastKnight() const;
```

- The methods **hasPaladinShield**, **hasLancelotSpear**, **hasGuinevereHair**, **hasExcaliburSword** return **true** if the Army of Knights respectively has the respective treasures: Paladin's Shield, Lancelot's Spear, Guinevere's Hair, and Excalibur's Sword. Otherwise, the return value is **false**. The declarations for these methods are as follows:

```
bool hasPaladinShield() const;
bool hasLancelotSpear() const;
bool hasGuinevereHair() const;
bool hasExcaliburSword() const;
```

5.4 Items

The class **BaseItem** is an abstract class representing information for an item. The class has two public pure virtual methods:

- **canUse**

```
virtual bool canUse(BaseKnight * knight) = 0;
```

  The method returns `true` if the knight can use the item, otherwise returns `false`.

- **use**

```
virtual void use(BaseKnight * knight) = 0;
```

The method of performing an action on the Knight is represented by the variable `knight` to change the Knight's parameters to suit the effect of the item.

Classes Antidote, PhoenixDownI, PhoenixDownII, PhoenixDownIII, PhoenixDownIV respectively represent the items: Antidote, PhoenixDown Type I, II, III, IV respectively. These classes inherit from the BaseItem class and must redefine the two methods canUse and use accordingly. For the Antidote class, students can define additional attributes for the Knight to represent the state of being poisoned.

5.5 Opponents

The class **BaseOpponent** is an abstract class that represents information for an opponent.

The classes **MadBear**, **Bandit**, **LordLupin**, **Elf**, **Troll**, **Tornbery**, **QueenOfCards**, **NinaDeRings**, **DurianGarden**, **OmegaWeapon**, **Hades** are classes representing opponents in events 1 through 10 respectively. These classes inherit from the **BaseOpponent** class. Students propose methods by themselves in these classes so that when Knight objects execute the **fight** method, the Knight's information is updated accordingly.

5.6 Bags

Each Knight will carry an item bag to store the items prepared before starting the journey and the items the knight picks up during the journey. The bag is implemented by a singly linked list. When an item is picked up, it will be placed at the top of the bag (also at the top of the singly linked list). When looking for an item, the knight will search from the beginning to the end of the bag. When he wants to use an item, the knight will exchange the needed item found with the item at the top of the bag, then remove the item to be used (now at the top of the bag) from the bag.

Besides, each type of knight uses a different bag:

- For the sake of combat, each Knight must not carry too many items. The Dragon Knight's bag will contain a maximum of 14 items. The maximum number for the Lancelot and the regular Knight of the Round Table is 16 and 19 items respectively. When a Knight's bag has reached the maximum number of items allowed, if the Knight picks up an item, the

Knight will give it to the Knight in front. This process continues until there is a knight who can put items in his bag. If the first Knight also fails to put the item in the bag, the item will be ignored.

The Dragon Knight is not affected by poison, so the Dragon Knight's bag does not contain the antidote. If the Knight picks up the antidote, the Dragon Knight will pass it to the Knight in front of him and the process continues as the item transfers above.

- Paladin's bag is enchanted by Merlin with space magic so there is no limit to the number of items.

The class **BaseBag** is a class representing the Item Bag. The class has one attribute **knight** of type **BaseKnight*** that represents the knight who owned this bag. The class has the following public methods:

- **insertFirst**

```
virtual bool insertFirst(BaseItem * item);
```

The method returns `true` if the item can be added to the bag, otherwise returns **false**. The way to add items to the bag is to add the item to the top of the linked list.

- **get**

```
virtual BaseItem * get(ItemType itemType);
```

The method returns a pointer to an object representing the first item in the bag of the same type as **itemType**. If there are no items of the same type, the value **NULL** is returned. When there is an item of the same itemType, the item will be used, then it is necessary to remove the item from the bag (which is a linked list). The way to delete an item is to exchange the item with the item at the top of the linked list, then delete the node at the top of the linked list. The way to find items is to search from the beginning of the linked list to the end of the linked list.

- toString()

```
virtual string toString() const;
```

The method returns a string representing information for the BaseItem object. The string representation has the following format:

```
Bag[count=<c>;<list\_items>]
```

In which:

  - ¡c¿: is the current number of items in the bag.

- ¡list_items¿: is a string representing the items from the beginning to the end of the linked list, each item is represented by the type name of the item, and the items are separated by a comma. The names of the item type are Antidote, PhoenixI, PhoenixII, PhoenixIII, PhoenixIV respectively.

An example of a string representation of a bag with 3 items in order from the beginning to the end of the list is Antidote, PhoenixDownIV, PhoenixDownI respectively.

```
Bag[count=3;Antidote,PhoenixIV,PhoenixI]
```

Students self-propose classes that inherit from the BaseBag class and represent the different types of bags of each type of Knight. The constructor of these bags takes 3 input parameters. The first parameter is of type **BaseKnight*** represents the Knight who owned this bag. The two next parameters are both of type integer, respectively, the number of phoenixdownI and the number of initial antidote Knight carried. Call these two numbers a and b respectively. If $a > 0$, the bag is initialized with a Nodes, which are PhoenixDowns of type I. Next, if $b > 0$, the bag is added to the b Nodes of Antidote at the top of the linked list. Note that when adding to the list, always add it at the beginning.

5.7 class KnightAdventure

Implement class **KnightAdventure** with methods:

- **loadArmyKnights** accepts a string parameter containing a file describing information for the Knights Army. Initialize the ArmyKnights object (which is a property of the class) from the provided file.
- **loadEvents** takes a string parameter containing a file describing information for the events. Initialize the Events object (which is a property of the class) from the provided file.
- **run** with no parameters, describing the process of the Knights Army going through events. At the end of each event, print out the information about the Army using printInfo() of the ArmyKnights class. After the last event, call the method **printResult** of ArmyKnights class with the argument value of **true** if the Knights Army wins Ultimecia, or **false** if the Army is defeated.

6 Requirements

To complete this assignment, students must:

1. Read entire this description file.
2. Download the initial.zip file and extract it. After extracting, students will receive files including main.cpp, main.h, knight2.h, knight2.cpp, and example file inputs. Students will only submit 2 files, knight2.h and knight2.cpp. Therefore, you are not allowed to modify the main.h file when testing the program.
3. Example testcases with output are posted on the submission site. Students run these examples by themselves. If their results are different from the submission site, the students will post the question and the results (with the details of each calculation step) on the Forum.
4. Students use the following command to compile:
   g++ -o main main.cpp knight2.cpp -I . -std=c++11
   Students use the following command to run the program:
   ./main tc1_armyknights tc1_events
   The above command is used in the command prompt/terminal to compile the program. If students use an IDE to run the program, students should pay attention: add all the files to the IDE's project/workspace; change the IDE's compile command accordingly. IDEs usually provide buttons for compiling (Build) and running the program (Run). When you click Build, the IDE will run a corresponding compile statement, normally, only main.cpp should be compiled. Students need to find a way to configure the IDE to change the compilation command, namely: add the file knight2.cpp, add the option -std=c++11, -I .
5. The program will be graded on the Unix platform. Students' backgrounds and compilers may differ from the actual grading place. The submission place on BKeL is set up to be the same as the actual grading place. Students must test the program on the submission site and must correct all the errors that occur at the BKeL submission site in order to get the correct results when final grading.
6. Modify the files knight2.h, knight2.cpp to complete this assignment and ensure the following two requirements:

   - Implement classes as described. Students can add more properties and other methods in the classes if necessary. All methods in this assignment description must be implemented for the compilation to be successful. If the student has not implemented a method yet, provide an empty implementation for that method.

- At the end of each event, students need to print out the Army information using the **printInfo()** method written in the ArmyKights class.
- Dynamically allocated memory in the heap memory must be freed before the program terminates.
- There is only one **include** directive in the knight2.h file, which is **#include "main.h"**, and one directive in the knight2.cpp file, which is **#include "knight2.h"**. Apart from the above directives, no other **#include** is allowed in these files.

7. Students are encouraged to write additional functions to complete this assignment.

7 Submission

Students submit only 2 files: knight2.h and knight2.cpp, before the deadline given in the link "Assignment 2 - Submission". There are a number of simple test cases used to check student work to ensure that student results are compilable and runnable. Students can submit as many times as they want, but only the final submission will be graded. Since the system cannot bear the load when too many students' submissions at once, students should submit their work as soon as possible. Students do so at their own risk if they submit assignments by the deadline. When the submission deadline is over, the system will close so students will not be able to submit any more. Submissions through other means will not be accepted.

8 Other regulations

- Students must complete this assignment on their own and must prevent others from stealing their results. Otherwise, the student is treated as cheating according to the regulations of the school for cheating.
- Any decision made by the teachers in charge of this assignment is the final decision.
- Students are not provided with test cases after grading, students will be provided with the assignment's score distribution.
- Assignment contents will be harmonized with a question in exam with similar content.

9 Handling fraud

The assignment must be done BY YOURSELF. Students will be considered fraudulent if:

- There is an unusual similarity between the source code of the submissions. In this case,

ALL submissions are considered fraudulent. Therefore, students must protect the source code of their assignments.

- Students do not understand the source code written by themselves, except for the parts of the code provided in the initialization program. Students can consult from any source, but make sure they understand the meaning of all the lines they write. In the case of not understanding the source code of the place they refer to, students are especially warned NOT to use this source code; instead, use what has been learned to write programs.
- Mistakenly submit another student's assignment on your personal account.
- Use code from ChatGPT.

In the case of cheating, students will get a 0 for the entire subject (not just the assignment).

DO NOT ACCEPT ANY INTERPRETATION AND NO EXCEPTION!

After the assignment has been submitted, some students will be called for random interviews to prove that the assignment has been done by themselves.

——————————END——————————