COMP 2012 Object-Oriented Programming and Data Structures

Assignment 3 Post Office 3.0

Photo credits: Martin Chan, SCMP

Introduction

The story

You were hired by the Hong Kong Post to revamp the postal processing system.

The new postal distribution system

HKP is looking to switch to a system where, at first, mail will be distributed from the Central Post Office to the main post offices located in every district. Once the mail reaches the district post offices (DPO), it will further be distributed to individual mailman - there are HASH_MODULO of them in each DPO. Each mailman further organizes their mail to easily search for them in their bags.

Remembering your good old days in HKUST's COMP 2012, you decided that the best way to sort the mails would be through Binary Search Trees.

The Mail Items

Each mail item will be unique - having different content, address, etc. Based on this, you decide a hashtable at each DPO to determine which mailman will be responsible for individual mail item.

Read the manuals

As a contractor for the Government, you need to be careful with reading the manuals and descriptions (otherwise you won't receive your pay). Please, do read all the description and examples carefully.

Read the FAQ page for some common clarifications. You should check that a day before the deadline to make sure you don't miss any clarification, even if you have already submitted your work.

If you need further clarification of the requirements, please feel free to post on the Piazza (via Canvas) with the pa3 tag. However, to avoid cluttering the forum with repeated/trivial questions, please do **read all the given code, webpage description, sample output, and the latest FAQ (refresh this page regularly) carefully before posting your questions**. Also, please be reminded that we won't debug any student's assignment for the sake of fairness.

Menu

Page maintained by

Yigit Sen
Email:
ysen@connect.ust.hk

Last Modified:
11/27/2022 15:12:18

Homepage

Course Homepage

Submission details are in the Submission and Deadline section.

We value academic integrity very highly. Please read the Honor Code section on our course webpage to make sure you understand what is considered as plagiarism and what the penalties are. The following are some of the highlights:

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters. We also review codes for potential cases manually.
- The penalty (for **BOTH** the copier and the copiee) is not just getting a zero in your assignment. Please read the Honor Code thoroughly.
- Serious offenders will fail the course immediately, and there may be additional disciplinary actions from the department and university, upto and including expulsion.

Download

- Skeleton code: skeleton.zip
- Sample Output & Grader Main: testcases.zip

Please download it now, as we will refer to it from time to time in the description below. Your task is to complete the missing implementation in the given skeleton code for 3 classes:

- **BST_Mailman** to be implemented in BST_Mailman.cpp
- **BST_PostOffice** to be implemented in BST_PostOffice.cpp
- **Mail** to be implemented in Mail.cpp

and submit them to ZINC in a zip file.

If you use VS Code, you may follow the creating a project and using the terminal for custom compilation command section on our VS Code usage tutorial. That is, create a folder to hold all the extracted files in your file explorer, then open this folder in VS Code. You can then use the terminal command `g++ -std=c++11 -o programName *.cpp` to compile all sources in the folder to the program. You are also welcome to create a Makefile for it yourself. After the compilation, you can then use the command `./programName` OR `.\programName` (depends on the OS/shell you use) to run the program.

Mail object

Before you do anything PA-related, it is important for you to study the objects that you will be sorting.

```
class Mail {
    private:
    //Each mail item will have a unique id
        const int id;
        const std::string address;
        const District district;
        const std::string streetName;
        const std::string content;
        int addressHash;
        void hashAddress();

    public:
        // Constructor for the Mail object
        Mail(const int id,const std::string address,const District distric
        const std::string streetName, const std::string content);

        // Getter functions
        // As a mail in real life is unmodifiable after
        // posting it, you won't need any setters.
        const int getId() const;
        const std::string &getAddress() const;
        const District getDistrict() const;
        const std::string &getContent() const;
        const int getAddressHash() const;
        const std::string getStreetName() const;

        // Prints the mail object stats
        void printMail() const;
};
```

This represents a mail that will go through the system.

Each mail object stores the district as an enum. The values possible are as follows:

```
enum District
  {
    Islands = 1,
    KwaiTsing,
    North,
    SaiKung,
    ShaTin,
    TaiPo,
    TsuenWan,
    TuenMun,
    YuenLong,
    Kowloon,
    KwunTong,
    ShamShuiPo,
    WongTaiSin,
    YauTsimMong,
    CentralAndWestern,
    Eastern,
    Southern,
    WanChai
  };
```

Tasks

You need to complete the implementation of the `hashAddress` function.
The following is the calculations that you need to use to implement hashAddress:
For each of the characters in the address, you multiply them by their location and sum it all up.
That is, for the address "hello"

```
h * 1 + e * 2 + l * 3 ...
```

You do the same for content, and add it all up to `addressHash` variable.

Tips

- The post office will use the hash value to split the mail to different mailman - study Mail.h file for an item that will help you to achieve this.
- Consider the limitations of the integer data type. After one point, it might end up with integer overflows. How can you overcome this?

BST_Mailman Class

Each mail carrier has a BST for themselves to organize their mail.

You may refer to the BST_Mailman.h while you read this section.

BST_Mailman.h is already complete. No modification should be made.

For this BST, you'll need to sort the mails based on the Street Name. The comparison operators are already implemented for strings, so you can use < and > to compare them.

In this PA, we'll follow a structure with a BST Node that holds the street name the mails are addressed to that street, and two pointers to left and right BST's. The BST only has a root pointer.

Data members in BST_Mailman_Node

```
std::string streetName;
```

Stores the street name that the mail is addressed to in that node.

```
Mail *mailPtr[MAX_NUM_MAILS]
```

Stores the mail pointers that is addressed to the `streetName`. Each Mailman will handle at most MAX_NUM_MAILS mails, and we will NEVER do a buffer overflow in this array.

```
int currentmailStored;
```

The current number of mail stored in the `mailPtr` - can be used as an upper boundary for for loops.

```
BST_Mailman *right, *left;
```

Pointers to the left and right BST's.

Data members in BST_Mailman

```
BST_Mailman_Node *root;
```

Pointer to the root node of the BST.

Member functions that you need to implement in BST_Mailman_Node

```
BST_Mailman_Node(Mail *mail)
```

Constructor for the BST_Mailman_Node. You need to make sure that you do proper memory management for mail objects (which can be tested in test cases).

```
~BST_Mailman_Node()
```

Destructor for the BST_Mailman_Node.

```
BST_Mailman * getLeftBST();
BST_Mailman * getRightBST();
```

Accessor functions for the BST's stored in the node.

```
 std::string getStreetName();
```

Accessor function for the street name.

```
void addMail(Mail * mail);
```

Adds mail to the `mailPtr` array (again, you may use currentmailStored here to see where to place that mail item.)

```
Mail *find(int id);
```

Finds the mail stored in the `mailPtr` array, returns the object if found, nullptr if not found.

```
bool remove(int id);
```

Removes a mail with the given `id`. Returns true if mail is found and removed, false if not found. TIP: Remove here also means that we have delivered the mail item and can remove them completely from the system.

Print function is given to you to ensure uniform outputs. In the BST_Mailman, you would have to implement the following functions:

```
BST_Mailman();
```

Constructor for the BST_Mailman

```
~BST_Mailman();
```

Destructor for the BST_Mailman.

```
void addMail(Mail *mail);
```

Add mail to the BST based on the street name of the mail.

```
Mail *find(int id, std::string streetName);
```

Based on the id and streetName given, find the mail object. Returns mail pointer if found, nullptr if not found.

```
bool remove(int id, std::string streetName);
```

Based on the id and streetName given, remove the mail object. Returns true if removed, false if not found.

```
void printInOrder();
void printPostOrder();
void printPreOrder();
```

Do the In-Order, Pre-Order and Post-Order prints.

BST_PostOffice Class

The CPO distributes the mail to DPO's through another BST.

You may refer to the BST_PostOffice.h while you read this section.

BST_PostOffice.h is already complete. No modification should be made.

For this BST, you'll need to sort the mails based on the District value.

Data members in BST_PostOffice_Node

```
District district;
```

Stores the districts that the mail is addressed to in that node.

```
BST_Mailman mailman[HASH_MODULO];
```

Each DPO has HASH_MODULO mail carriers employed, and mail is allocated to each carrier based on the hash value of the Mail, implemented in Mail.cpp.

```
BST_PostOffice *right, *left;
```

Pointers to the left and right BST's.

Data members in BST_PostOffice

```
BST_PostOffice_Node *root;
```

Pointer to the root node of the BST.

Member functions that you need to implement

```
BST_PostOffice_Node(Mail *mail)
```

Constructor for the BST_PostOffice_Node. You need to make sure that you do proper memory management for mail objects (which may be tested in test cases).

```
~BST_PostOffice_Node()
```

Destructor for the BST_PostOffice_Node.

```
BST_PostOffice * getLeftBST(); BST_PostOffice * getRightBST();
```

Accessor functions for the BST's stored in the node.

```
std::string getDistrict();
```

Accessor function for the district value.

```
void addMail(Mail * mail);
```

Adds mail to the appropriate mail carrier's BST.

```
Mail *find(int id, std::string streetName);
```

Finds the mail stored in the DPO, returns the object if found, nullptr if not found. TIP: Check each mail carrier's BST.

```
bool remove(int id, std::string streetName);
```

Removes a mail with the given `id`. Returns true if mail is found and removed, false if not found.

```
 void print(int type);
```

Prints the mail in each mail carrier's box based on the type given.

Type here means 1 = inorder, 2 = postorder, 3 = preorder traversal

```
 void printMailman(int i, printType type);
```

Prints the mail in the mail carrier # i, based on the given type, and i = index value in the array.

For printType, please refer to the following enum:

```
enum printType
{
    inorder = 1,
    postorder = 2,
    preorder = 3
};
```

In the BST_PostOffice, you would have to implement the following functions:

```
BST_PostOffice();
```

Constructor for the BST_PostOffice

```
~BST_PostOffice();
```

Destructor for the BST_PostOffice.

```
void addMail(Mail *mail);
```

Add mail to the system.

```
Mail *find(District dist, int id, std::string streetName);
```

Based on the district, id and streetName given, find the mail object.

```
bool remove(District district, int id, std::string streetName);
```

Based on the district, id and streetName given, remove the mail object.

```
void printInOrder();
void printPostOrder();
void printPreOrder();
```

Do the In-Order, Pre-Order and Post-Order prints.

```
 void printMailman(District district, int i, printType type);
```

Prints the mail in the mail carrier # i at the district given, based on the type given; i = index value in the array.

For printType, please refer to the following enum:

```
enum printType
{
    inorder = 1,
    postorder = 2,
    preorder = 3
};
```

```
void printDistrict(District district, printType type);
```

Given the district, print all of the mail items at that DPO.

Tips & Tricks

- You might want to call BST_Mailman's functions inside BST_PostOffice
- Each class requires different ways to manage mail objects.
- Read .h files and make sure you understand what kinds of data structures you have access to.

BST Structure & Key Variables

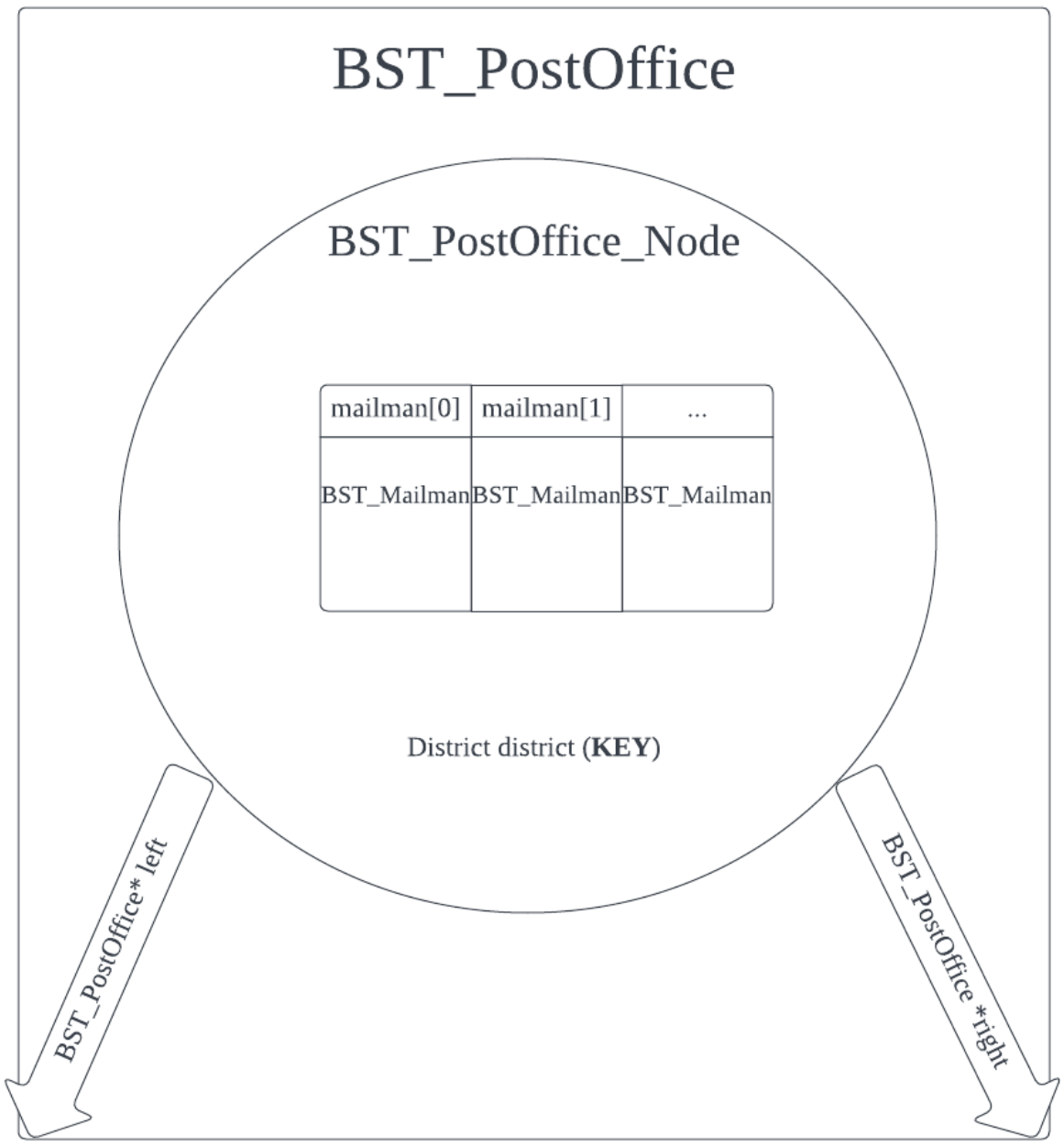
In this PA, you have 4 different key values that you need to keep a track of:

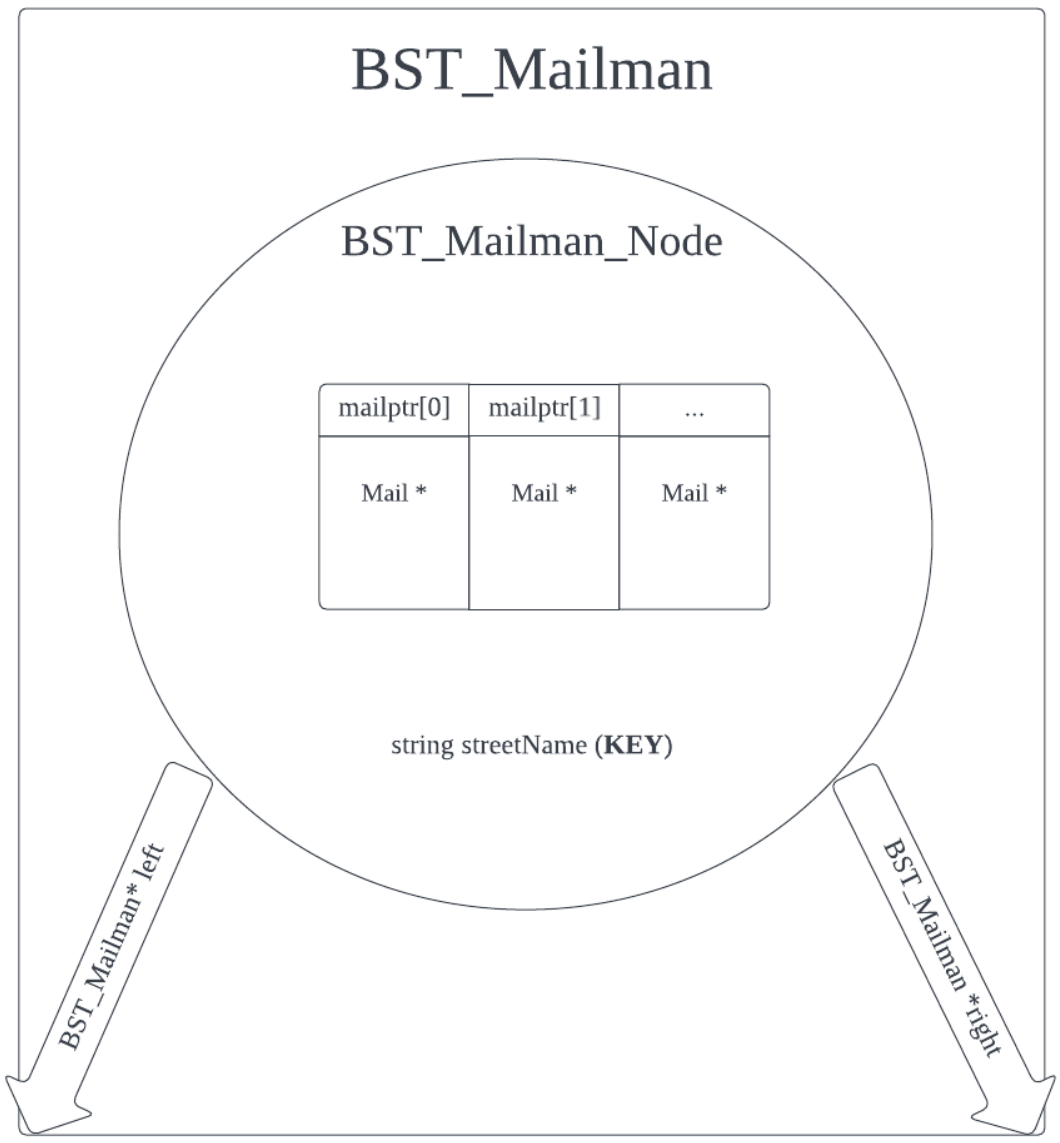
- **Mail::id** is an unique identifier of the mail object. No two mail objects will share the same id.
- **District** is the key used in BST_PostOffice to distribute mails to individual District Post Offices.
- **Mail::addressHash** is being used to determine which mailman is responsible with the delivery of the mail.
- **Mail::streetName** is the key used in BST_Mailman to sort mails.

Each mail goes through the following distirbution mechanism.

- Mail gets routed to District Post Offices through BST_PostOffice based on their District value.
- In each district, each mailman (i.e. mailman[0] to mailman[HASH_MODULO-1]) represents a single mailman. Mail arriving into DPO's then gets allocated to mailman based on their addressHash.
- Each mailman then sorts their mails based on the streetName of the mail objects.

You may refer to the following diagrams to understand how the BST structure works.





Sample Output and Grading Scheme

Sample output and executables:

- Test cases: testcases.zip
- Windows: pa3.exe
- Mac (Intel): pa3_intel_mac.exe
- Mac (ARM): pa3_arm_mac.exe

About memory leak and other potential errors
Memory leak checking is done via the `-fsanitize=address,leak,undefined` option (related documentation here) of a recent g++ compiler on Linux (it won't work on Windows for the versions we have tested). Check the "Errors" tab (next to "Your Output" tab in the test case details popup) for errors such as memory leaks. Other errors/bugs such as out-of-bounds, use-after-free bugs, and some undefined-behavior-related bugs may also be detected. You will get 0 mark for the test case if there is any error there. Note that if your program has no errors detected by the sanitizers, then the "Errors" tab may not appear. If you wish to check for memory leaks yourself using the same options, you may follow our Checking for memory leaks yourself guide.

After the deadline
We will have 34 additional test cases, which won't be revealed to you before the deadline. Together with the 34 given test cases, there will then be 68 test cases used to give you the final assignment grade. All 68 test cases will be run two times as well: once without memory leak checking and once with memory leak checking. Note that, for this assignment, we will give a higher score weighting to the hidden test cases. The weighting will be determined later, but all the hidden cases combined will definitely weigh more than all the given ones combined. Additionally, if we found you had hardcoded your solution for the inputs in the given cases, a penalty would be applied. Details will be provided in the marking scheme, which will be released after the deadline.

Submission and Deadline

Deadline: 23:59:00 on November 30th, 2022 (Wednesday)

Please zip and submit the following 3 files.

- Mail.cpp
- BST_PostOffice.cpp
- BST_Mailman.cpp

Submit the zip file (no RAR, 7z, etc.) to ZINC. ZINC usage instructions can be found here.

Notes:

- You may submit your file multiple times, but only the last submission will be graded. **You do NOT get to choose which version we grade.** If you submit after the deadline, late penalty will be applied according to the submission time of your last submission.
- Submit early to avoid any last-minute problems. Only ZINC submissions will be accepted.
- The ZINC server will be very busy in the last few days, especially in the last few hours, so you should expect you would get the grading result report not-very-quickly. However, as long as your submission is successful (i.e. you see "[Your name] submitted on [Some day] at [Some time]" without any error in the log), we will grade your latest submission with all test cases after the deadline.
- In the grading report, pay attention to various errors reported. For example, **under the "make" section, if you see a red cross, click on the STDERR tab to see the compilation errors.** You must fix those before seeing any program output for the test cases below.
- Make sure you submit the correct file yourself. You can download your file back from ZINC to verify. Again, **we only grade what you uploaded last to ZINC**.

About any ZINC server / webpage / UI problem

If you see a blank page or some errors after some action on ZINC, you may try refreshing the page and repeating the action, and it usually will work the second/third time. If it still doesn't work, try using another browser / private window. Please also email the details about the bug you encountered (potentially with a screenshot if that is helpful) to zinc@cse.ust.hk , and they will try to fix the bug as soon as possible.

Again, it is not unusual that reports may not be generated until hours later when the deadline approaches. When the ZINC cannot give you the grading report quickly, please test your program on your computer, and you may compare your output with the sample output we provided on the PA webpage yourself. Please do NOT resubmit the same file again and again, as it will lag the server and won't make it give you the report faster.

While you may not get the grading report quickly, as long as your submission is successful (the grading part and submission collection part are working separately on ZINC, and the submission collection part is usually responsive and stable), we will grade your latest submission with all test cases after the deadline, so don't worry too much.

If testing on a Linux machine (on which ZINC is operating) is needed for some reason, you may refer to the previous section for instructions on remote-accessing a Linux lab machine. It is not exactly the ZINC server, but it is close in many scenarios, meaning you may be able to reproduce the crash/problems there.

Compilation Requirement

It is **required** that your submissions can be compiled and run successfully in our online auto-grader ZINC. If we cannot even compile your work, it won't be graded. Therefore, for parts you cannot finish, just put in a dummy implementation so that your whole program can be compiled for ZINC to grade the other parts you have done. Empty implementations can be like the following:

```
int SomeClass::SomeFunctionICannotFinishRightNow()
{
    return 0;
}

void SomeClass::SomeFunctionICannotFinishRightNowButIWantOtherPartsGraded(
{
}
```

Late submission policy

There will be a penalty of -1 point (out of a maximum of 100 points) for every minute you are late. For instance, since the deadline for the assignment is 23:59:00 on Nov 30th, submitting your solution at 1:00:00 on Dec 1st, will result in a penalty of -61 points for your assignment. However, the lowest grade you may get from an assignment is zero: any negative score after the deduction due to a a late penalty (and any other penalties) will be reset to zero.

FAQ

Frequently Asked Questions

Please also check the FAQ & Errata page on Piazza for questions related to PA3.

Q: My code doesn't work / there is an error. Here is the code. Can you help me fix it?

A: As the assignment is a major course assessment, to be fair, you are supposed to work on it on your own, and we should not finish the tasks for you. We might provide some very general hints , but we shall not fix the problem or debug the problem for you.

Q: Can I add extra helper functions?

A: You may do so in the files that you can modify and submit. That implies you cannot add new member functions to any given class.

Q: Can I include additional libraries?

A: No. Everything you need is already included - there is no need for you to add any include statement (under our official environment).

Q: Can I use global variables or static variables such as "static int x"?

A: No.

Q: Can I use "auto"?

A: No.

Q: Can I use "goto"?

A: No.

Q: Can I use function X or class Y in this assignment?

A: In general, if it is not forbidden in the description and the previous FAQs, and you can use it without including any additional library on ZINC, then you can use it. We suggest quickly testing it on ZINC (to see if a basic usage of it compiles there) before committing to using it, as library inclusion requirements may differ in different environments.

Q: My program gives the correct output on my computer, but it gives a different one on ZINC. What may be the cause?

A: Usually inconsistent strange result (on different machines/platforms, or even different runs on the same machine) is due to relying on uninitialized hence garbage values, missing return statements, accessing out-of-bound array elements, improper use of dynamic memory, or relying on library functions that might be implemented differently on different platforms (such as pow() in cmath).
You may find a list of common causes and tips on debugging in the notes here.
In this particular PA, it is probably related to the misuse of dynamic memory. Good luck with bug hunting!