Final Report

for Program Design 2

By Team15【提姆】

數學四 趙曉峰 408210054

資工一 李和豫 411410021

傳播一 郭宏哲 411335004

資工三 谷品儒 407410042

**Menu**

[Introduction 2](#_Toc137461873)

[Program Design 2](#_Toc137461874)

[Basic Part 3](#_Toc137461875)

[Advanced Part 15](#_Toc137461876)

[Demonstration 18](#_Toc137461877)

## Introduction

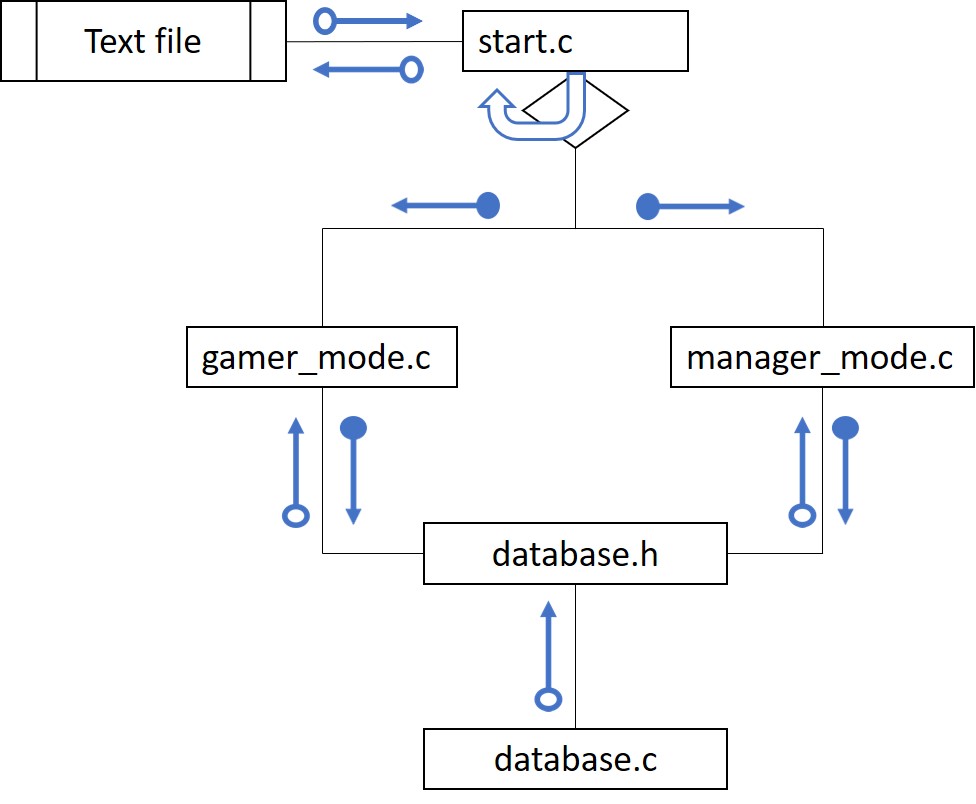
This is a Quiz game like “Kahoot!” or “Quiz Ranger”, which was popular in 2010s. We built question database and two user modes, including “Gamer” and “Manager”; user can select one of them to enter the system.

Select “Gamer” mode and enjoy the simple pleasure! Cautiously answer ten questions and try your best to get the highest score. If you have made some mistakes, don”t worry… You could always strive again and again!

Moreover, we welcome all imaginative creator to shine your talent! Join “Manager” mode to add, search, and edit questions. We have completed guide to help you create your unique question bank, step by step!

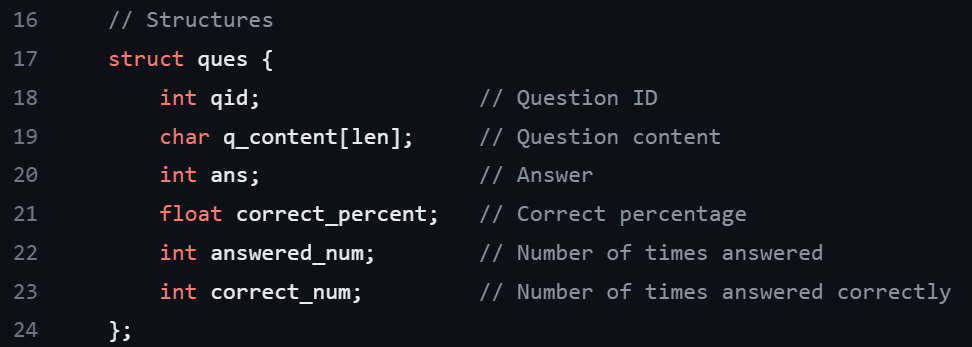
Now, join and enjoy your question time!

## Program Design



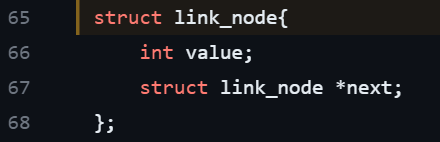
## Basic Part

#### Data Type and Data Structure

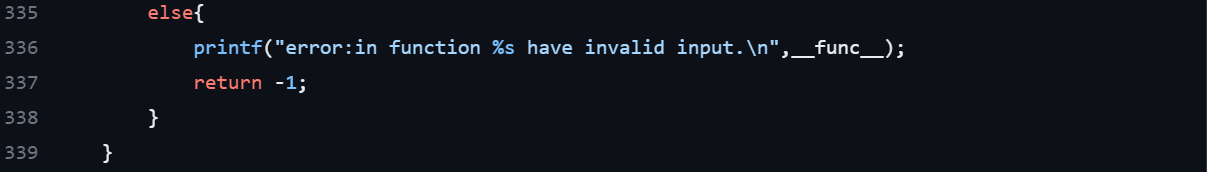
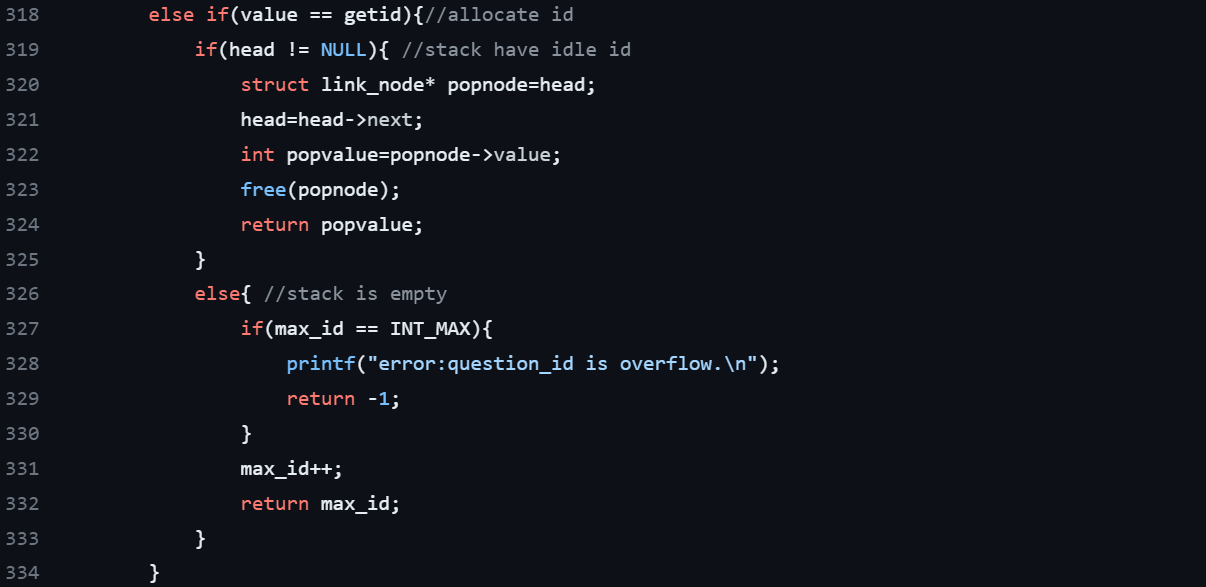
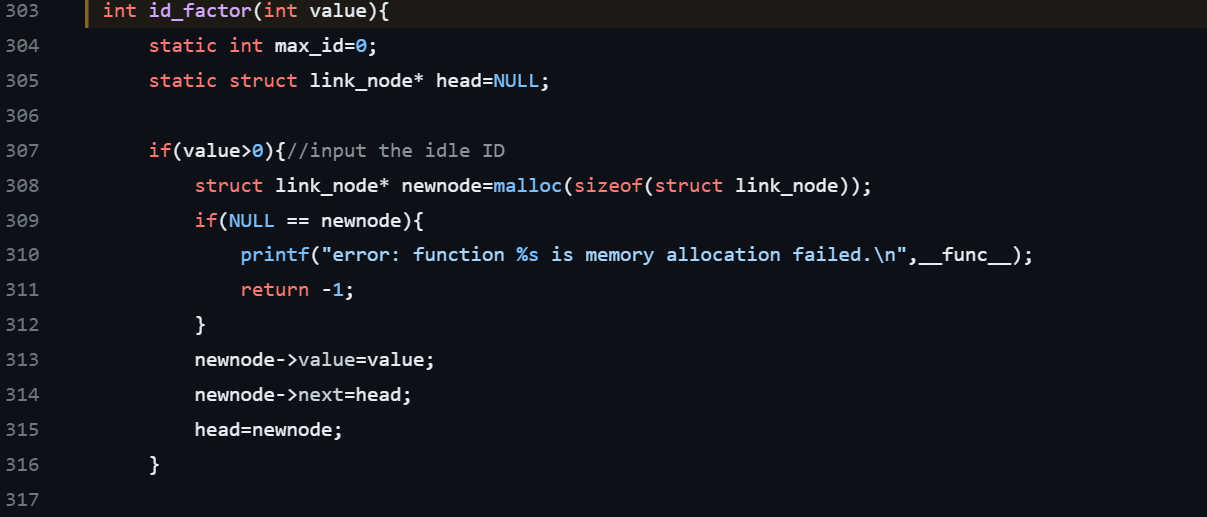
 A unit of question in the database includes one multiplechoice question, one answer, and a pass rate. It should be noticed that the options are included in question, which is represented by “**q\_content[len]**”. Please take a look at the structure code below, which written in the file “**database.h**”.

1. int
2. “qid”: question id, assign a number to each question for managing it.
3. “ans”: the answer of question.
4. “answered\_num”: count the answeringtime of the question. This data is recorded for computing pass rate.
5. “correct\_num”: count the times of correctanswering. This data is recorded for computing pass rate.
6. float
7. “correct\_percent”: pass rate of the question. The data is computed with the equation, “correct\_percent” = “correct\_num” ÷ “answered\_num”. After each gamer”s response, the success rate will be permanently recorded in the system.
8. char & string
9. “q\_content[len]”: content of question and option. The limit of characters is 499.
10. linked list

Structure “**link\_node**” is defined in file “**database.c**”.



The linked list function “**id\_factor**” is declared in file “**database.c**”.



The function “**id\_factor**” in the provided code is responsible for generating and managing unique question IDs (“**qid**”). It manages the allocation and recycling of question IDs, ensuring they are unique and within the appropriate range.

Here”s a breakdown of what it does:

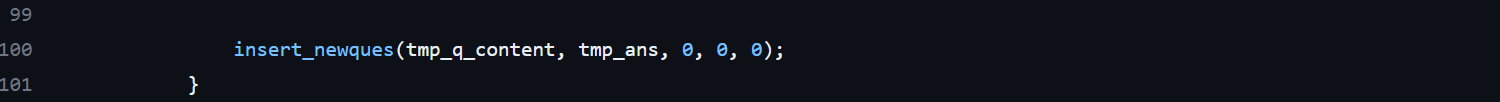
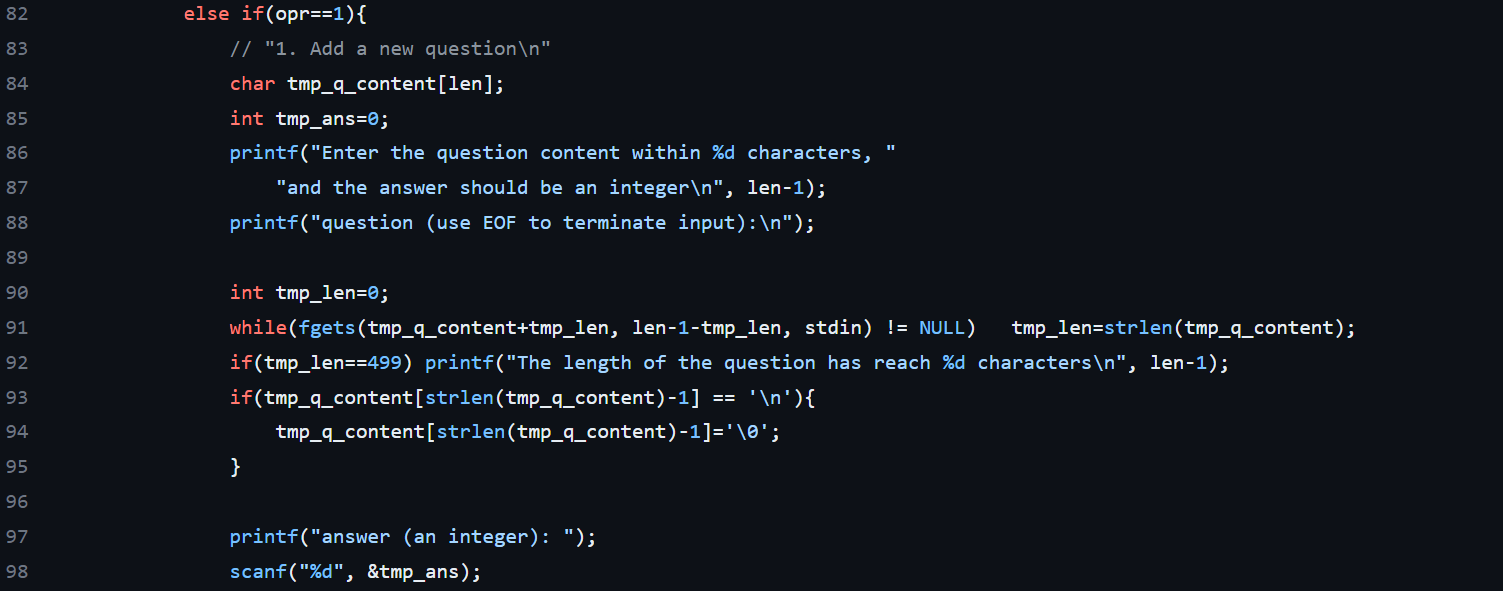
1. The function takes an integer value as input. If the input value is greater than 0, it means an idle ID is being provided to the function, and it creates a new node in a linked list called head to store this idle ID. Containing line 307 ~ 316, it appropriates in function “**delete\_ques**”.
2. If the input value is equal to “getid” (which is defined as 2100483647), it means the function needs to allocate a new question ID. In this case, the function checks if there are any idle IDs in the linked list. If there are, it pops the topmost idle ID from the list and returns it. If the linked list is empty, it increments the “max\_id” (which is defined as macro “INT\_MAX”) variable and returns the new value as the allocated question ID. In line 318 ~ 334, it uses both stack and linked list to complete it.
3. If the input value does not fall into the above two cases, it means an invalid input was provided, and an error message is displayed.

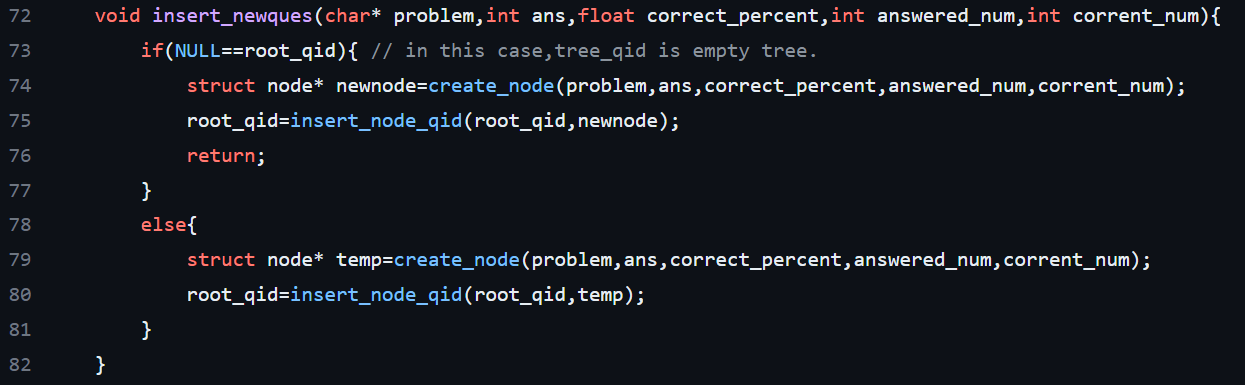
#### Operations

1. Add

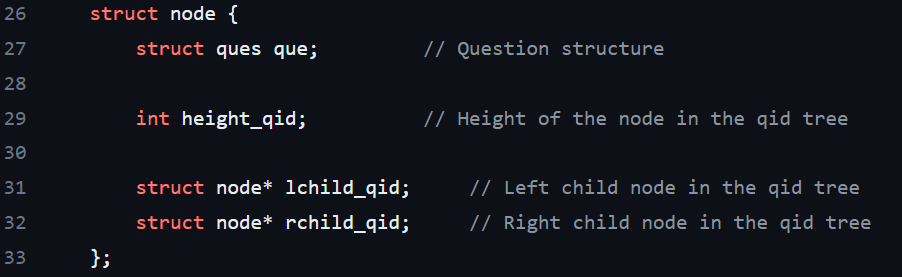
|  |  |
| --- | --- |
| *File* | *Function* |
| manager\_mode.c | manager\_mode |
| database.c | insert\_newques  create\_node  insert\_node\_qid |

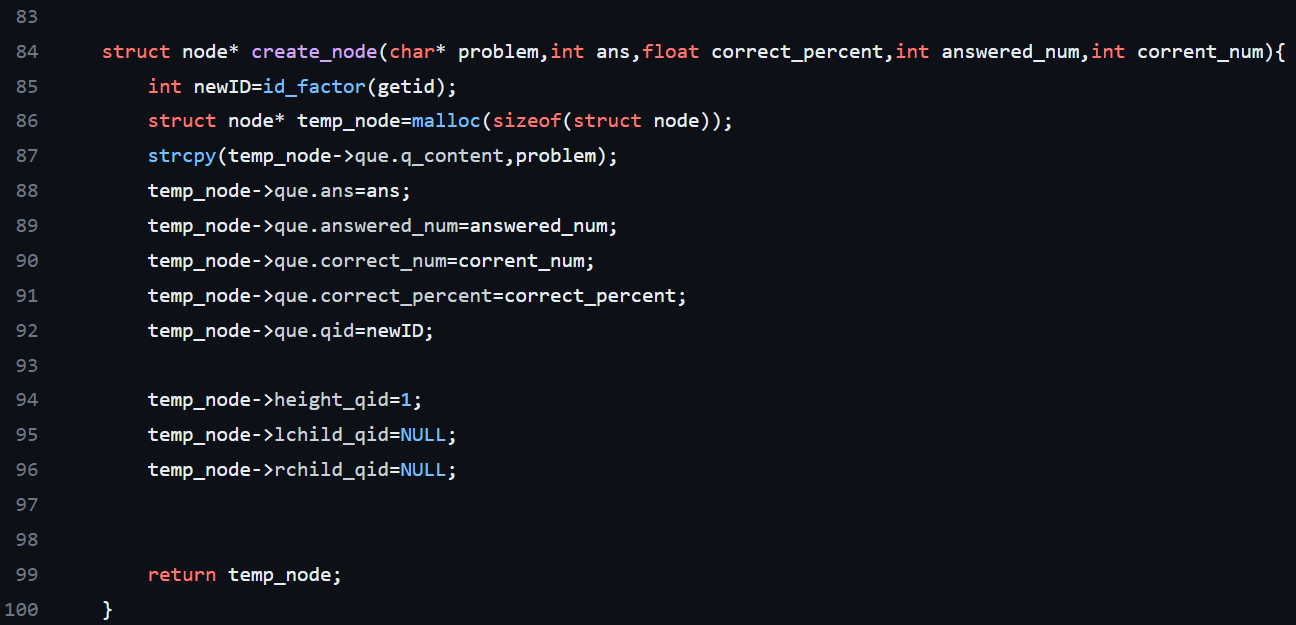
The first picture is the code in file “**manager\_mode.c**”. This part receives user”s input and call the function “**insert\_newques**” (which declared in “**database.h**”) to store the data into database.



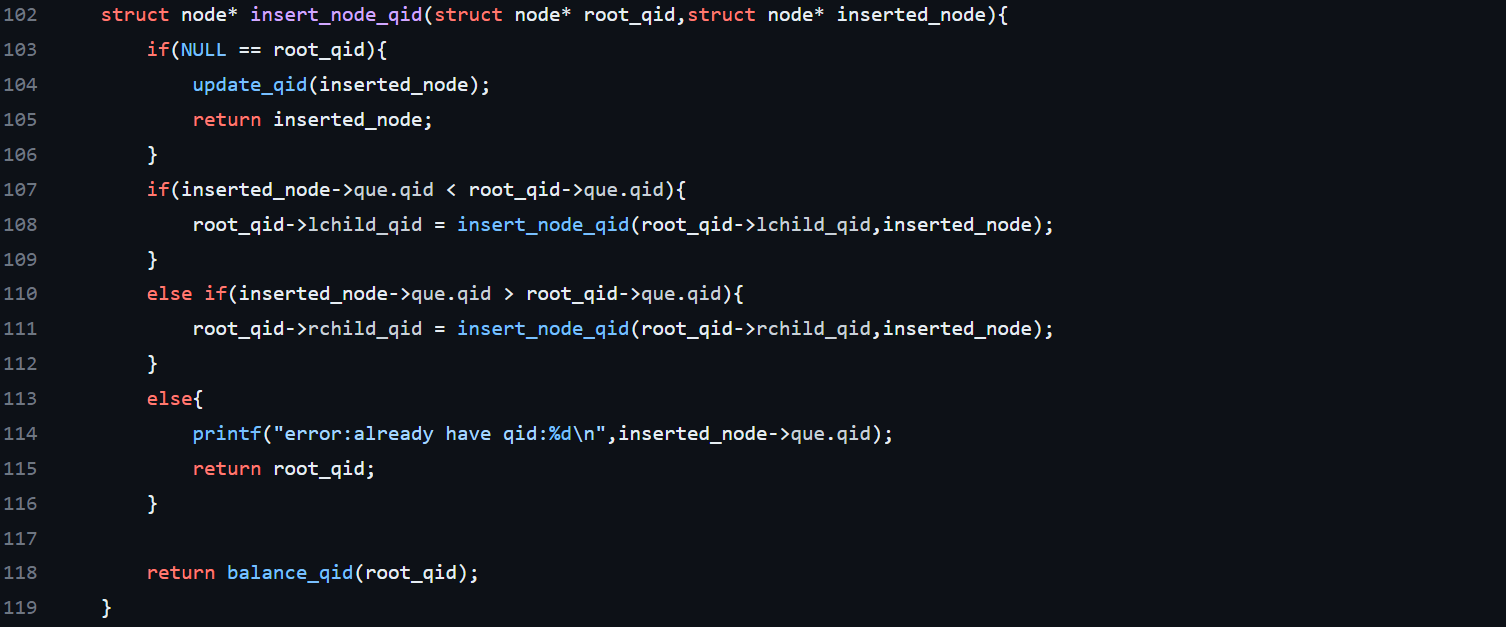
 The code below is the function “**insert\_newques**”, which defined in file “**database.c**”. Actually, apart from using linked list for assigning question IDs, the system primarily uses AVL tree to organize data because that is much more efficient than linked list. Although AVL tree is the category of advanced part, it will still be mentioned here since the entire system is composed of tree structure.

Before representing the other two functions in it, please take a look at the struct “node” below, which is the definition of node of tree in system.

 The node contains the struct “ques” (i.e., the struct has been displayed in p.3), “\*height\_qid” for balancing, “\*lchild\_qid”, and “\*rchild\_qid”.

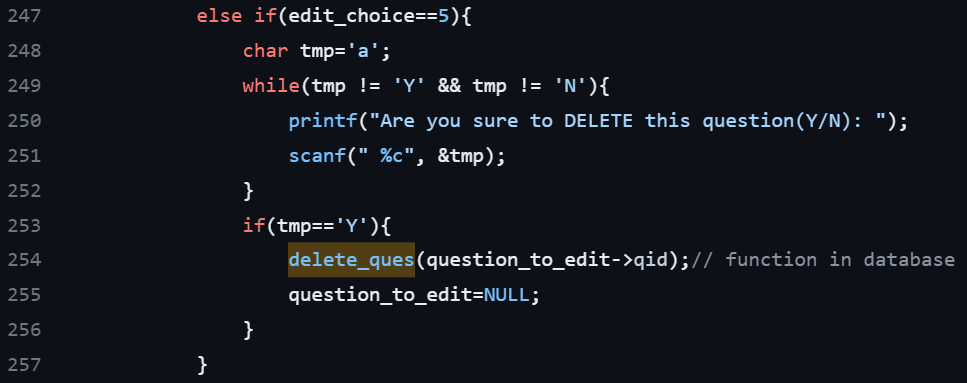
 Let see the function “**create\_node**”. Its main effect is to create a new node containing new data for tree.

About the function “**insert\_node\_qid**”, involving to AVL tree, the detail will be explained in advanced part. In this part, the code in green frame is the error message.

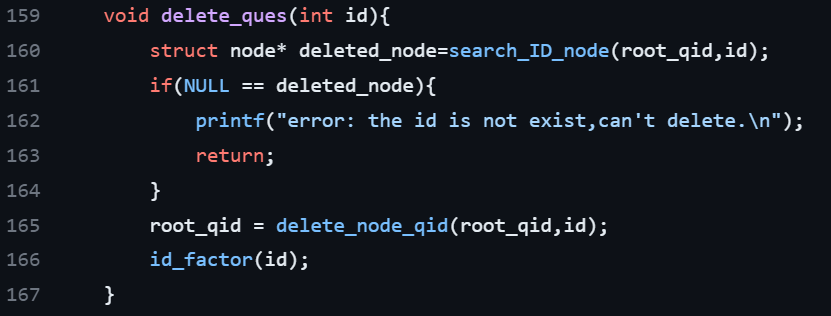


1. Delete

|  |  |
| --- | --- |
| *File* | *Function* |
| manager\_mode.c | manager\_mode  edit\_question\_infomation |
| database.c | delete\_ques  delete\_node\_qid |

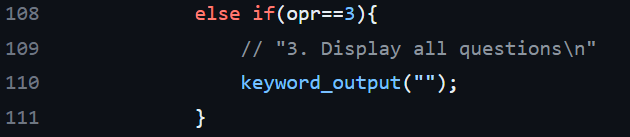
 The first picture is the code in file “**manager\_mode.c**”. It faces to user and calls the function “**delete\_ques**” to adjust the database.

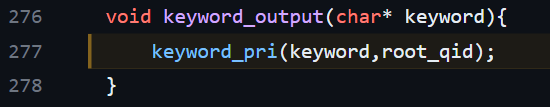
The next picture is the function “**delete\_ques**” in file “**database.c**”. The error message is in the green frame.

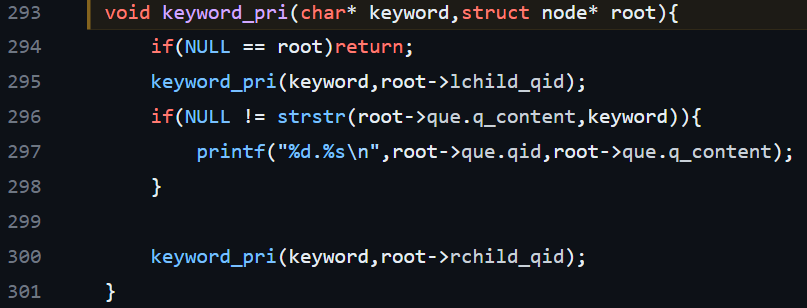


1. Traverse

|  |  |
| --- | --- |
| *File* | *Function* |
| manager\_mode.c | manager\_mode |
| database.c | keyword\_output  keyword\_pri |

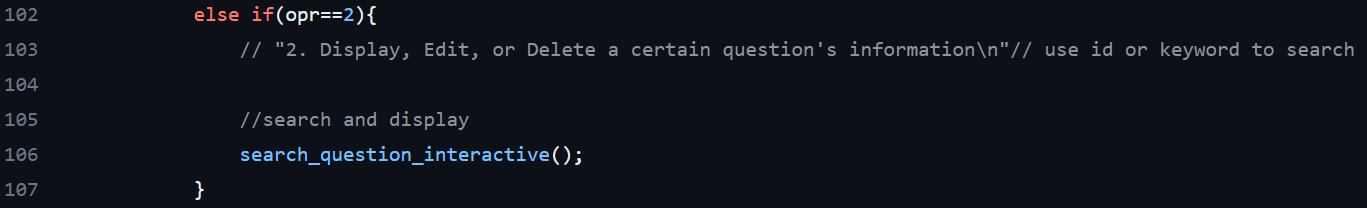
 The picture below is the code in file “**manager\_mode.c**”. It faces to user and call the function “**keyword\_output**”, which defined in the file “**database.c**”.

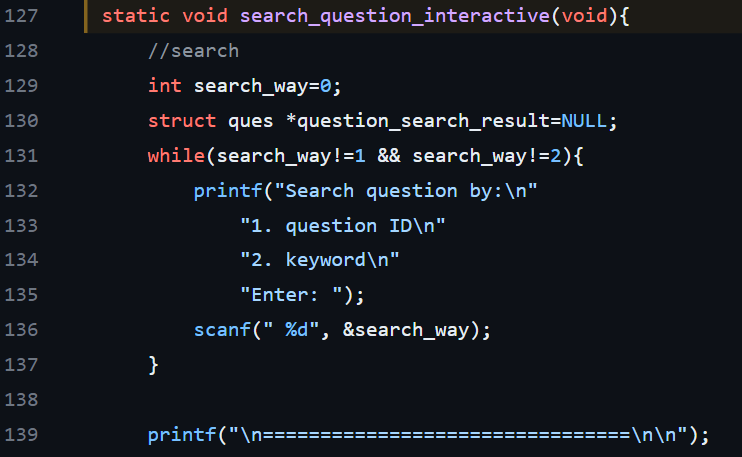
 Because the function “**keyword\_output**” is designed for printing all question contains the specific keyword, so the value here sets empty. Then, it calls the function “**keyword\_pri**”, which contains a node parameter “root\_qid” of tree.

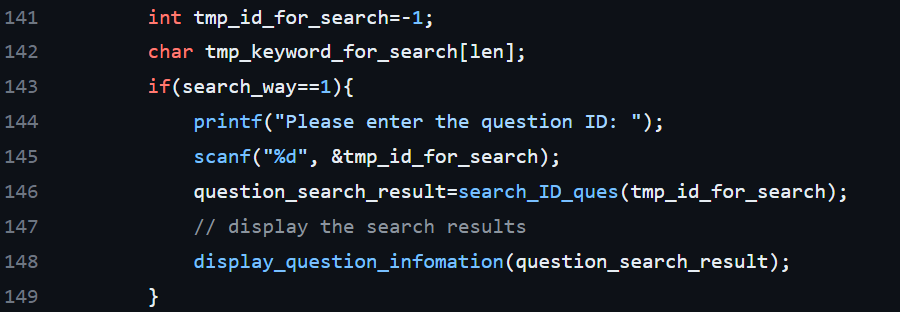
 The main effect of the function “**keyword\_pri**” is to print all the content of the question unit from the root of tree.

1. Search

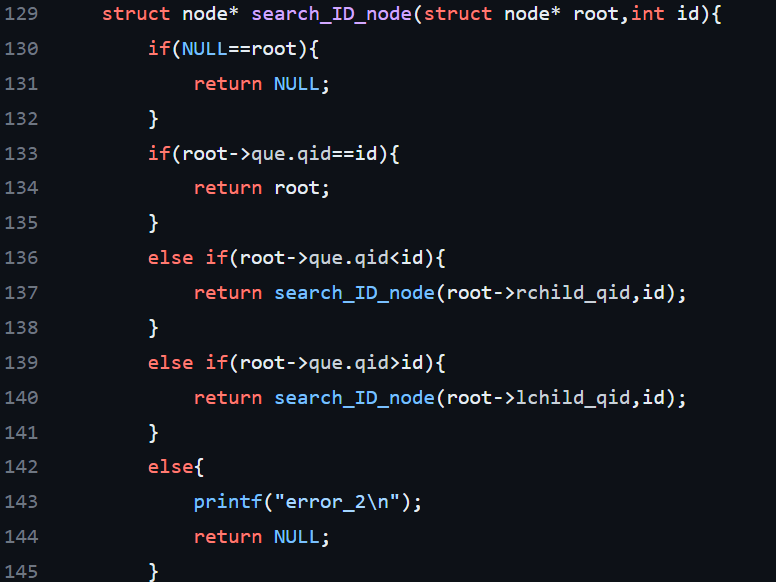
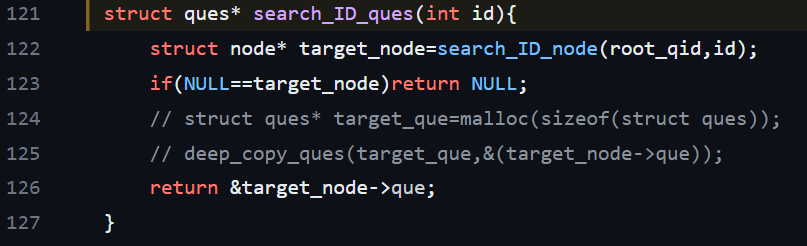
|  |  |
| --- | --- |
| *File* | *Function* |
| manager\_mode.c | manager\_mode  search\_question\_interactive  display\_question\_infomation |
| database.c | search\_ID\_ques  keyword\_output |

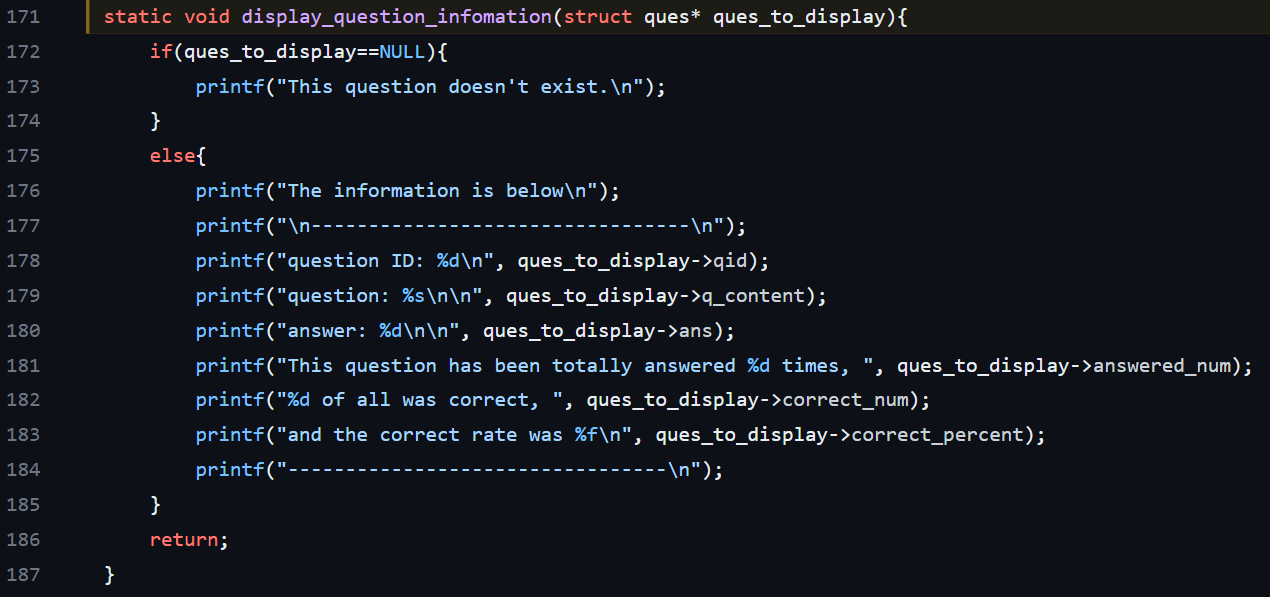
The first picture is the code in file “**manager\_mode.c**”.

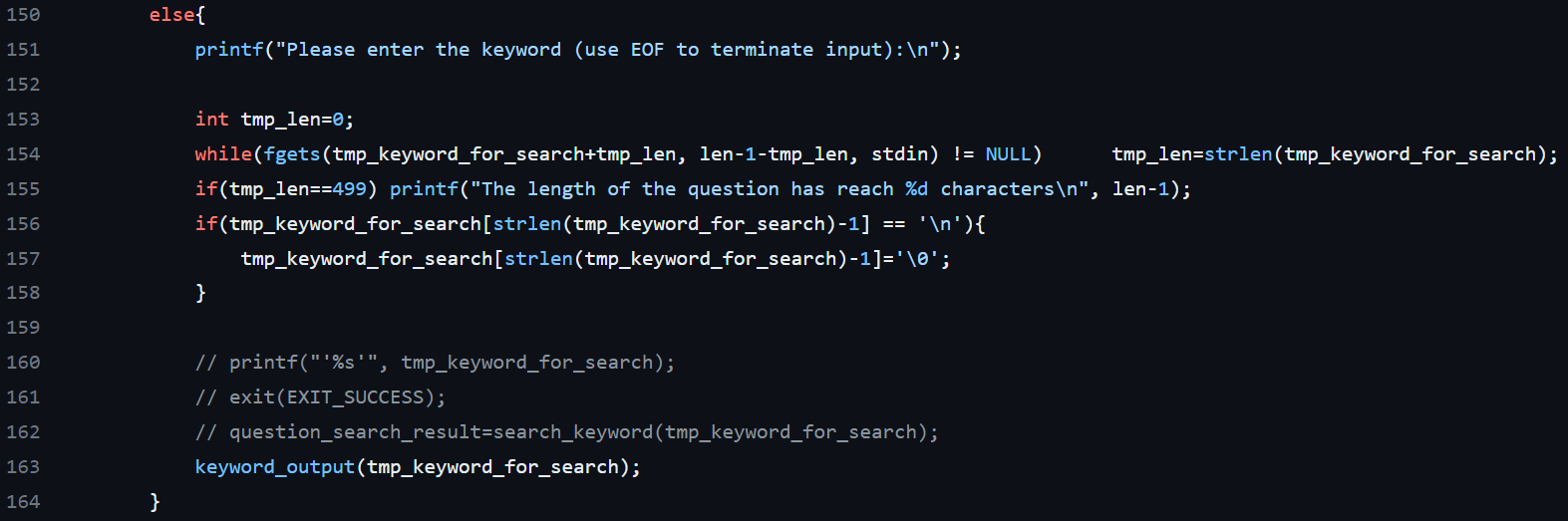
 The designer designed two ways to search the specific question. One uses question ID, the other uses keyword.

1. The one that using question ID calls the function “**search\_ID\_ques**”, which is defined in the file “**database.c**”.

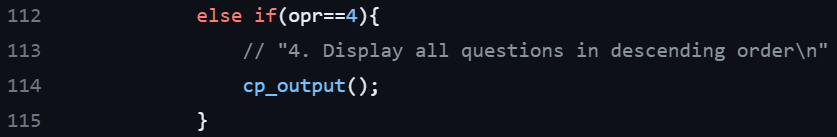
The code below represents the function “**search\_ID\_ques**” and its relative function “**search\_ID\_node**” in the file “**database.c**”.

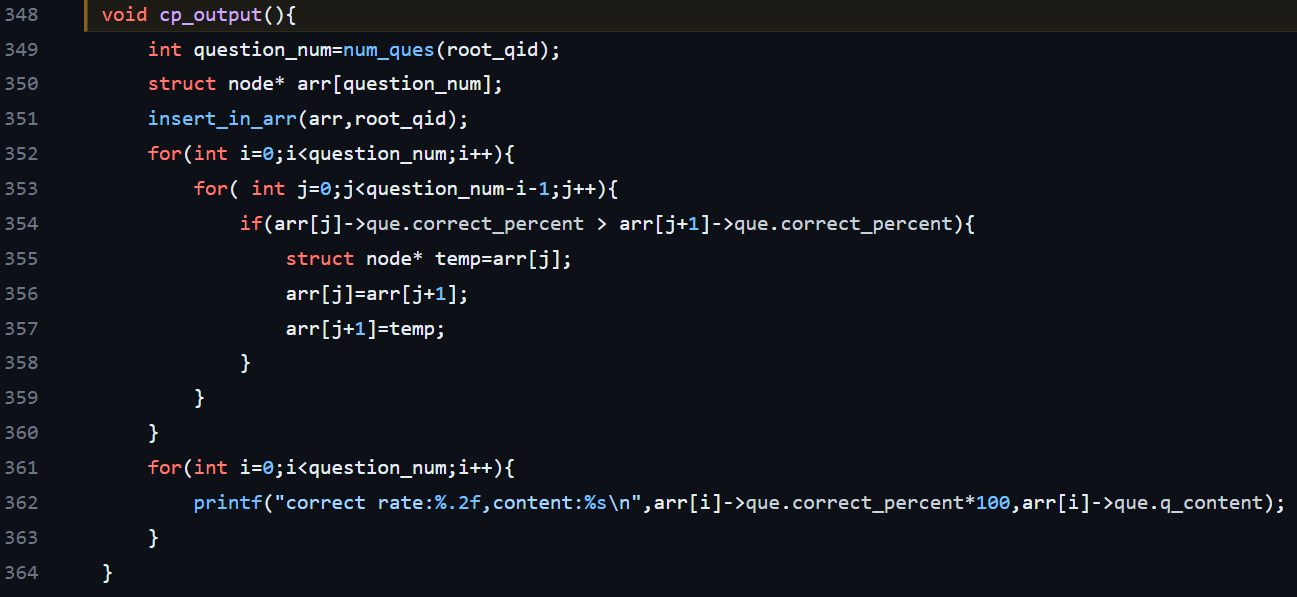


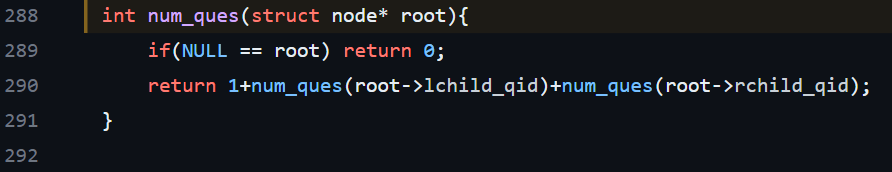
The function at line 148 in file “**manager.c**”, “**display\_question\_infomation**”, is written for printing the result of “search\_ID\_ques” returning.

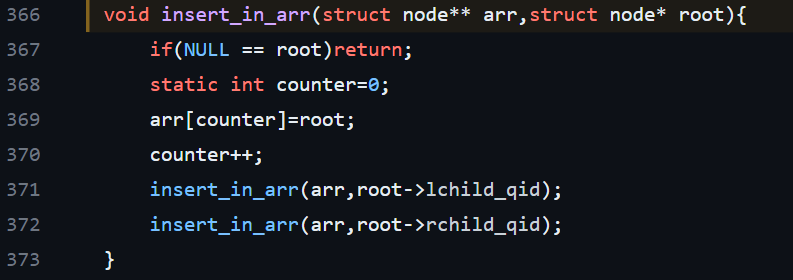
1. The other way, using keyword to search the specific question in database, is the code below in file “**manager\_mode.c**”. It calls the function “**keyword\_output**” in the file “**database.c**”, the value here sets specific keyword. (p.s. The screenshot of code is placed at Traverse part.) The function “**keyword\_output**” will print content of all the questions containing specific keyword.
2. Sort

|  |  |
| --- | --- |
| *File* | *Function* |
| manager\_mode.c | manager\_mode |
| database.c | cp\_output  num\_ques  insert\_in\_arr |

 The code below is in file “**manager\_mode.c**”.

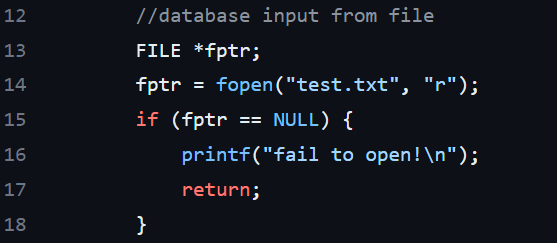
 The function “**cp\_output**”, which defined in the file “**database.c**”, is designed for displaying all question in ascending order.

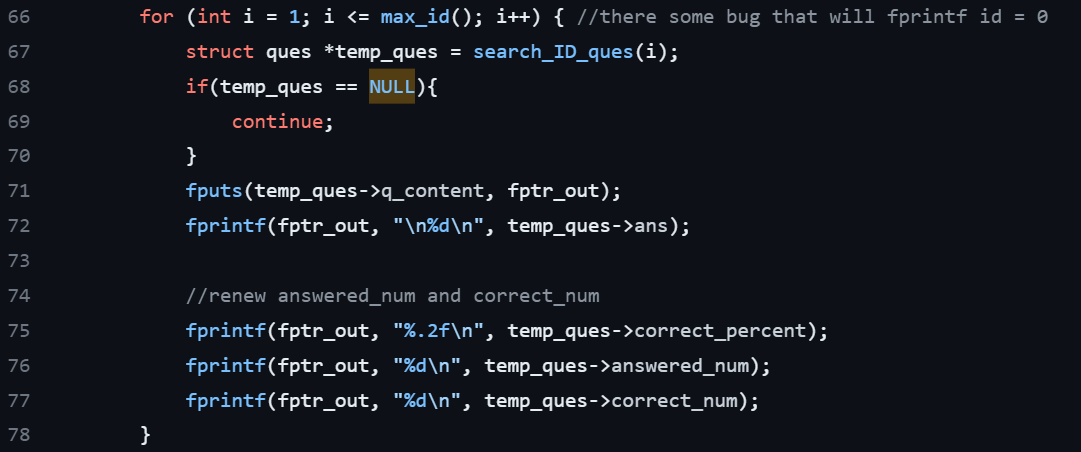
 The function “**num\_ques**” counts all the node in tree (i.e., the total number of questions).

 The function “**insert\_in\_arr**” is to traverse the AVL tree in a depthfirst manner and insert each node into the array. The array is used to store the nodes of the tree so that they can be processed or accessed in order.

After inserting all the nodes, function “**cp\_output**” will sort the array with bubble sort.

1. File I/O

 The database is built of text file, so both input and output files need to be in a specified format of text files. The following code, which written in the file “**start.c**”, shows how the system reads input file.

 The code below, which also written in the file “**start.c**”, shows how the system output the file after processed.

## Advanced Part

#### AVL tree

In the file “**database.c**”, the AVL tree is implemented to store the questions. The AVL tree is used to maintain the questions sorted by their “qid” (question ID).

The AVL tree related functions and structures in the code include:

1. Structure Definition:

“**struct node**”: Represents a node in the AVL tree, containing information about a question. (p.7)

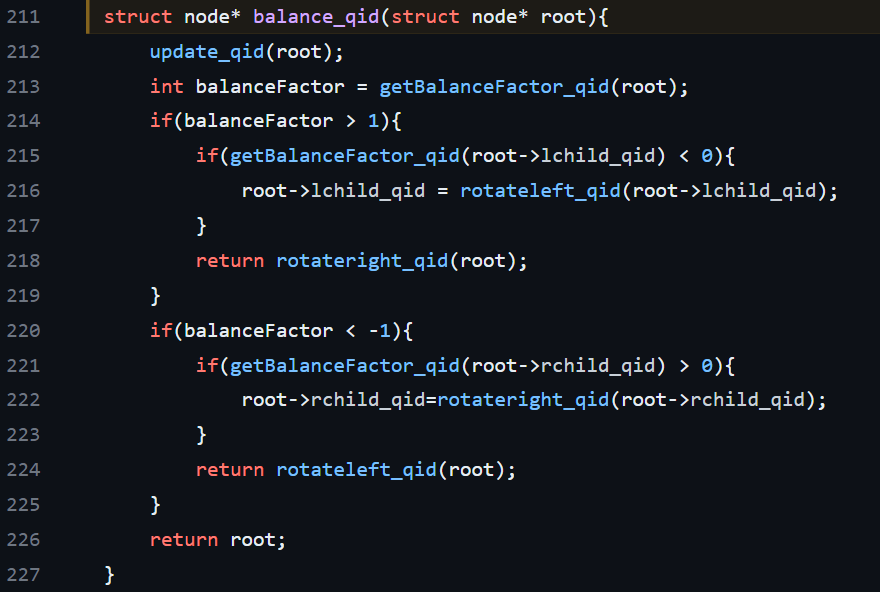
2. AVL Tree Operations:

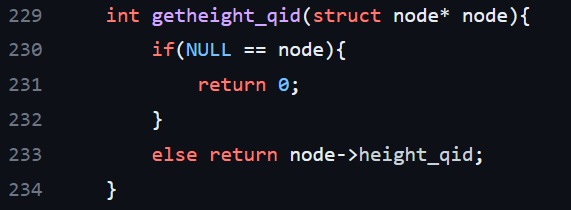
“**insert\_newques**”: Inserts a new question node into the AVL tree based on the “qid”. (p.6)

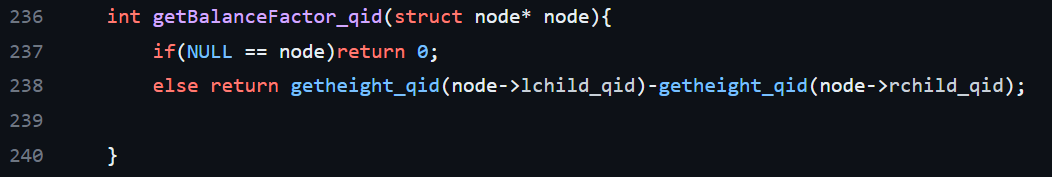
“**delete\_ques**”: Deletes a question node from the AVL tree based on the “qid”. (p.9)

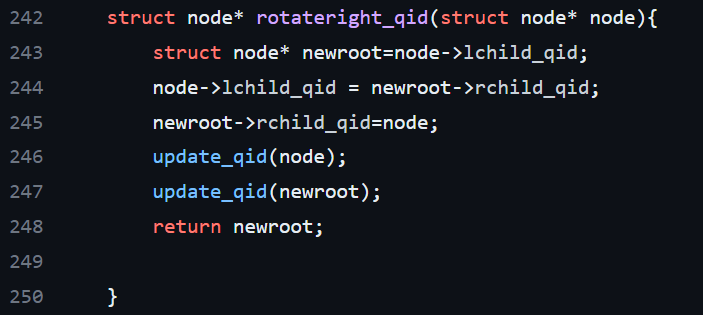
“**search\_ID\_ques**”: Searches for a question node in the AVL tree based on the “qid”. (p.11)

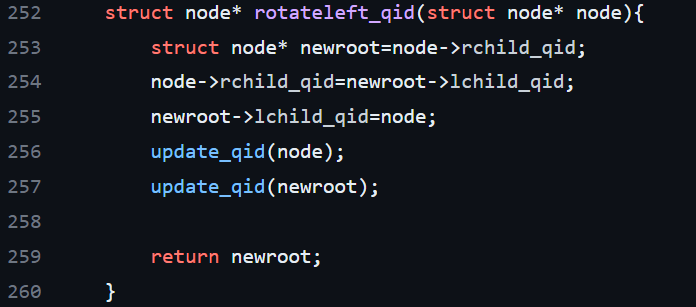
3. AVL Tree Balancing:

“**balance\_qid**”: Balances the AVL tree to maintain its height balance property after insertion or deletion operations.

 “**getheight\_qid**”: Calculates the height of a node in the AVL tree.

 “**getBalanceFactor\_qid**”: Calculates the balance factor of a node in the AVL tree.

 “**rotateright\_qid**”: Performs a right rotation on a node to balance the AVL tree.

 “**rotateleft\_qid**”: Performs a left rotation on a node to balance the AVL tree.

The “root\_qid” variable represents the root of the AVL tree.

To summarize, the AVL tree in the code is used to store the questions based on their “qid” and provides efficient insertion, deletion, and search operations while maintaining the balance of the tree.

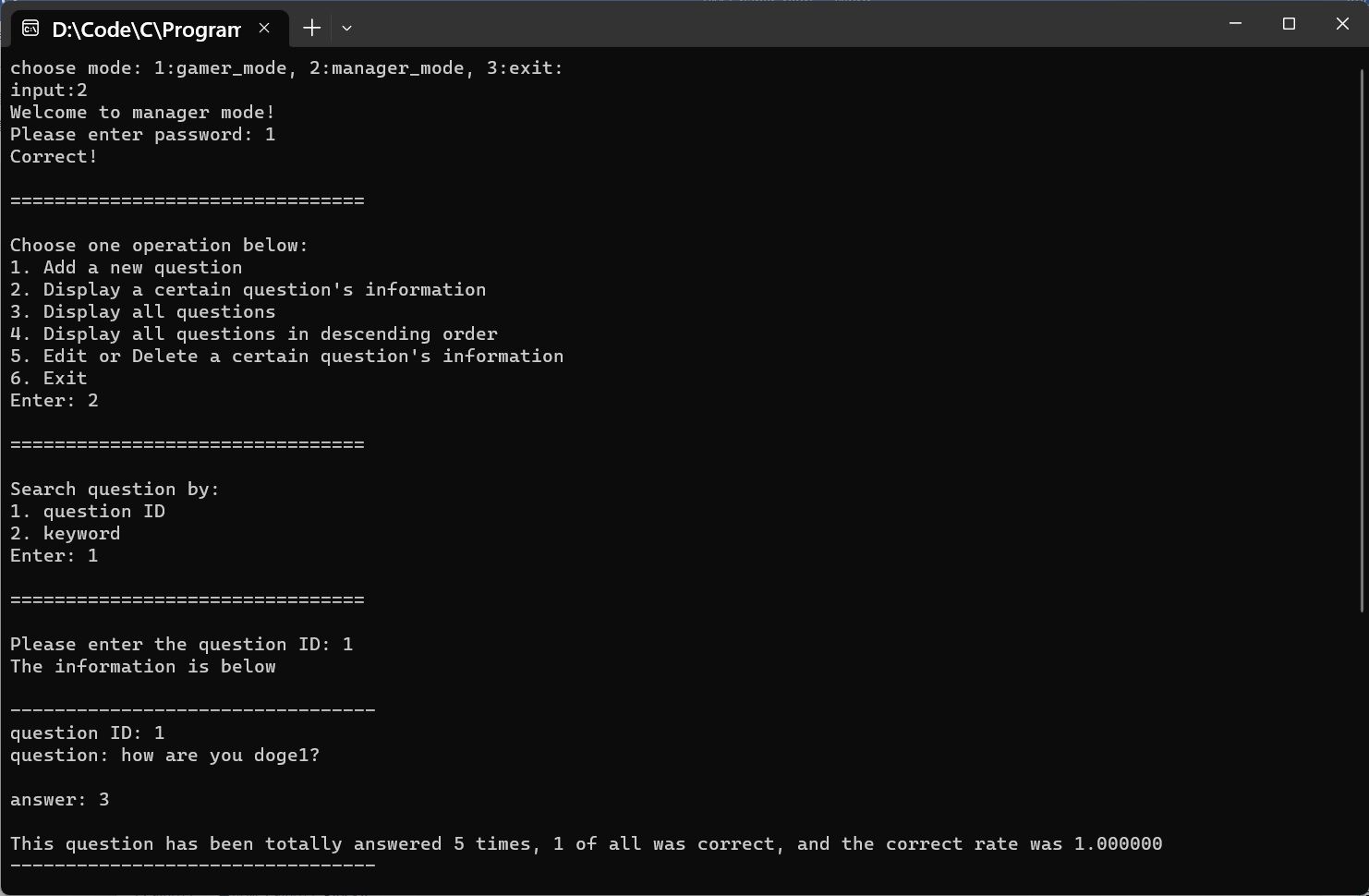
#### Search algorithm based on advanced structure

Which is detailed described at p.10~12.

#### Sort algorithm based on advanced structure

Which is detailed described at p.12~14.

#### User Interface

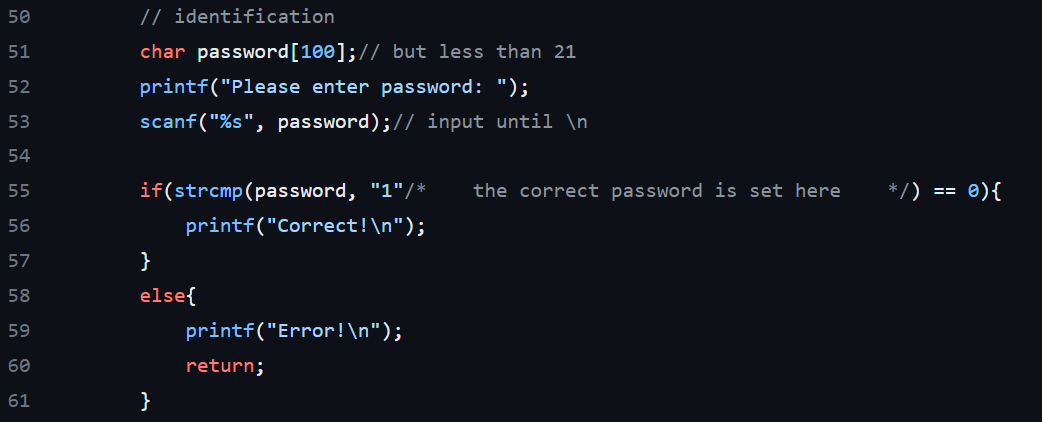
 The file “**gamer\_mode.c**” and “**manager\_mode.c**” output content is carefully formatted to guide the user step-by-step in using the system, making the entire system easy to understand and use. For example, this is a screenshot during system running:

#### Another design: Gamer mode

The team has designed a game mode to implement the functionality of the question bank and has established its theme as a quiz game. The completed content is written in the file “**gamer\_mode.c**”.

 In the gamer mode, the system will select ten questions randomly in question bank. It shows the result after user answers each time, and settle the total points in the end. An attentive inspiration is the game will show the message to encourage user if he/she keeps answering correctly.

#### Another design: Manager Password

 The code below is written in the file “**manager\_mode.c**”. The password is “1”.

## Demonstration

The link to the **GitHub repository**:

<https://github.com/AngelaKu123/111-2-PD-Final>