

# 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
- Program Design .....2
- Basic Part.....3
- Advanced Part .....15
- Demonstration .....18

# Introduction

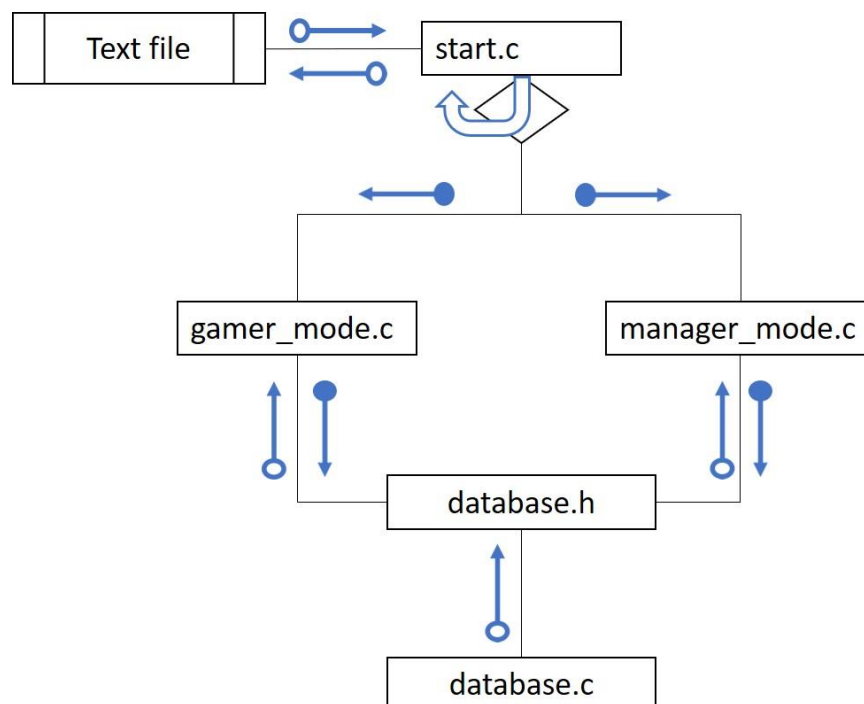
This is a Quiz game like “Kahoot!” or “Quiz Ranger”, which was popular in 2010s. We built question database and two user modes, including “Gamer” and “Manager”; user can select one of them to enter the system.

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**”.

```
16 // Structures
17 struct ques {
18     int qid;           // Question ID
19     char q_content[len]; // Question content
20     int ans;           // Answer
21     float correct_percent; // Correct percentage
22     int answered_num;   // Number of times answered
23     int correct_num;    // Number of times answered correctly
24 };
```

### 1. int

- (1) “qid”: question id, assign a number to each question for managing it.
- (2) “ans”: the answer of question.
- (3) “answered\_num”: count the answeringtime of the question. This data is recorded for computing pass rate.
- (4) “correct\_num”: count the times of correctanswering. This data is recorded for computing pass rate.

### 2. float

- (1) “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.

### 3. char & string

(1) "q\_content[len]": content of question and option. The limit of characters is 499.

#### 4. linked list

Structure "link\_node" is defined in file "database.c".

```
65     struct link_node{
66         int value;
67         struct link_node *next;
68     };
```

The linked list function "id\_factor" is declared in file "database.c".

```
303     int id_factor(int value){
304         static int max_id=0;
305         static struct link_node* head=NULL;
306
307         if(value>0){//input the idle ID
308             struct link_node* newnode=malloc(sizeof(struct link_node));
309             if(NULL == newnode){
310                 printf("error: function %s is memory allocation failed.\n",__func__);
311                 return -1;
312             }
313             newnode->value=value;
314             newnode->next=head;
315             head=newnode;
316         }
317
318         else if(value == getid){//allocate id
319             if(head != NULL){ //stack have idle id
320                 struct link_node* popnode=head;
321                 head=head->next;
322                 int popvalue=popnode->value;
323                 free(popnode);
324                 return popvalue;
325             }
326             else{ //stack is empty
327                 if(max_id == INT_MAX){
328                     printf("error:question_id is overflow.\n");
329                     return -1;
330                 }
331                 max_id++;
332                 return max_id;
333             }
334         }
335         else{
336             printf("error:in function %s have invalid input.\n",__func__);
337             return -1;
338         }
339     }
```

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

<i>File</i>	<i>Function</i>
<b>manager_mode.c</b>	manager_mode
<b>database.c</b>	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.

```

82     else if(opr==1){
83         // "1. Add a new question\n"
84         char tmp_q_content[100];
85         int tmp_ans=0;
86         printf("Enter the question content within %d characters, "
87             "and the answer should be an integer\n", 100);
88         printf("question (use EOF to terminate input):\n");
89
90         int tmp_len=0;
91         while(fgets(tmp_q_content+tmp_len, 100-tmp_len, stdin) != NULL) tmp_len=strlen(tmp_q_content);
92         if(tmp_len==100) printf("The length of the question has reach %d characters\n", 100);
93         if(tmp_q_content[strlen(tmp_q_content)-1] == '\n'){
94             tmp_q_content[strlen(tmp_q_content)-1]='\0';
95         }
96
97         printf("answer (an integer): ");
98         scanf("%d", &tmp_ans);
99
100        insert_newques(tmp_q_content, tmp_ans, 0, 0, 0);
101    }

```

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.

```

72 void insert_newques(char* problem,int ans,float correct_percent,int answered_num,int corrent_num){
73     if(NULL==root_qid){ // in this case,tree_qid is empty tree.
74         struct node* newnode=create_node(problem,ans,correct_percent,answered_num,corrent_num);
75         root_qid=insert_node_qid(root_qid,newnode);
76         return;
77     }
78     else{
79         struct node* temp=create_node(problem,ans,correct_percent,answered_num,corrent_num);
80         root_qid=insert_node_qid(root_qid,temp);
81     }
82 }

```

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”.

```
26     struct node {
27         struct ques que;           // Question structure
28
29         int height_qid;           // Height of the node in the qid tree
30
31         struct node* lchild_qid;   // Left child node in the qid tree
32         struct node* rchild_qid;   // Right child node in the qid tree
33     };
```

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

```
83
84     struct node* create_node(char* problem,int ans,float correct_percent,int answered_num,int corrent_num){
85         int newID=id_factor(getid);
86         struct node* temp_node=malloc(sizeof(struct node));
87         strcpy(temp_node->que.q_content,problem);
88         temp_node->que.ans=ans;
89         temp_node->que.answered_num=answered_num;
90         temp_node->que.correct_num=corrent_num;
91         temp_node->que.correct_percent=correct_percent;
92         temp_node->que.qid=newID;
93
94         temp_node->height_qid=1;
95         temp_node->lchild_qid=NULL;
96         temp_node->rchild_qid=NULL;
97
98
99         return temp_node;
100     }
```

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.



```

102     struct node* insert_node_qid(struct node* root_qid, struct node* inserted_node){
103         if(NULL == root_qid){
104             update_qid(inserted_node);
105             return inserted_node;
106         }
107         if(inserted_node->que.qid < root_qid->que.qid){
108             root_qid->lchild_qid = insert_node_qid(root_qid->lchild_qid, inserted_node);
109         }
110         else if(inserted_node->que.qid > root_qid->que.qid){
111             root_qid->rchild_qid = insert_node_qid(root_qid->rchild_qid, inserted_node);
112         }
113         else{
114             printf("error: already have qid:%d\n", inserted_node->que.qid);
115             return root_qid;
116         }
117
118         return balance_qid(root_qid);
119     }

```

## 2. 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.

```

247         else if(edit_choice==5){
248             char tmp='a';
249             while(tmp != 'Y' && tmp != 'N'){
250                 printf("Are you sure to DELETE this question(Y/N): ");
251                 scanf(" %c", &tmp);
252             }
253             if(tmp=='Y'){
254                 delete_ques(question_to_edit->qid); // function in database
255                 question_to_edit=NULL;
256             }
257         }

```

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

```
159 void delete_ques(int id){
160     struct node* deleted_node=search_ID_node(root_qid,id);
161     if(NULL == deleted_node){
162         printf("error: the id is not exist,can't delete.\n");
163         return;
164     }
165     root_qid = delete_node_qid(root_qid,id);
166     id_factor(id);
167 }
```

### 3. 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**”.

```
108 else if(opr==3){
109     // "3. Display all questions\n"
110     keyword_output("");
111 }
```

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.

```
276 void keyword_output(char* keyword){
277     keyword_pri(keyword,root_qid);
278 }
```

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

```

293 void keyword_pri(char* keyword,struct node* root){
294     if(NULL == root)return;
295     keyword_pri(keyword,root->lchild_qid);
296     if(NULL != strstr(root->que.q_content,keyword)){
297         printf("%d.%s\n",root->que.qid,root->que.q_content);
298     }
299
300     keyword_pri(keyword,root->rchild_qid);
301 }

```

#### 4. 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**”.

```

102     else if(opr==2){
103         // "2. Display, Edit, or Delete a certain question's information\n" use id or keyword to search
104
105         //search and display
106         search_question_interactive();
107     }

```

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

```

127 static void search_question_interactive(void){
128     //search
129     int search_way=0;
130     struct ques *question_search_result=NULL;
131     while(search_way!=1 && search_way!=2){
132         printf("Search question by:\n"
133             "1. question ID\n"
134             "2. keyword\n"
135             "Enter: ");
136         scanf(" %d", &search_way);
137     }
138
139     printf("\n===== \n\n");

```

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

```
141     int tmp_id_for_search=-1;
142     char tmp_keyword_for_search[len];
143     if(search_way==1){
144         printf("Please enter the question ID: ");
145         scanf("%d", &tmp_id_for_search);
146         question_search_result=search_ID_ques(tmp_id_for_search);
147         // display the search results
148         display_question_infomation(question_search_result);
149     }
```

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

```
121     struct ques* search_ID_ques(int id){
122         struct node* target_node=search_ID_node(root_qid,id);
123         if(NULL==target_node)return NULL;
124         // struct ques* target_que=malloc(sizeof(struct ques));
125         // deep_copy_ques(target_que,&(target_node->que));
126         return &target_node->que;
127     }
128
129     struct node* search_ID_node(struct node* root,int id){
130         if(NULL==root){
131             return NULL;
132         }
133         if(root->que.qid==id){
134             return root;
135         }
136         else if(root->que.qid<id){
137             return search_ID_node(root->rchild_qid,id);
138         }
139         else if(root->que.qid>id){
140             return search_ID_node(root->lchild_qid,id);
141         }
142         else{
143             printf("error_2\n");
144             return NULL;
145         }
```

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

```

171 static void display_question_infomation(struct ques* ques_to_display){
172     if(ques_to_display==NULL){
173         printf("This question doesn't exist.\n");
174     }
175     else{
176         printf("The information is below\n");
177         printf("\n-----\n");
178         printf("question ID: %d\n", ques_to_display->qid);
179         printf("question: %s\n\n", ques_to_display->q_content);
180         printf("answer: %d\n\n", ques_to_display->ans);
181         printf("This question has been totally answered %d times, ", ques_to_display->answered_num);
182         printf("%d of all was correct, ", ques_to_display->correct_num);
183         printf("and the correct rate was %f\n", ques_to_display->correct_percent);
184         printf("-----\n");
185     }
186     return;
187 }

```

- (2) 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.

```

150 else{
151     printf("Please enter the keyword (use EOF to terminate input):\n");
152
153     int tmp_len=0;
154     while(fgets(tmp_keyword_for_search+tmp_len, len-1-tmp_len, stdin) != NULL)    tmp_len=strlen(tmp_keyword_for_search);
155     if(tmp_len==499) printf("The length of the question has reach %d characters\n", len-1);
156     if(tmp_keyword_for_search[strlen(tmp_keyword_for_search)-1] == '\n'){
157         tmp_keyword_for_search[strlen(tmp_keyword_for_search)-1]='\0';
158     }
159
160     // printf("%s", tmp_keyword_for_search);
161     // exit(EXIT_SUCCESS);
162     // question_search_result=search_keyword(tmp_keyword_for_search);
163     keyword_output(tmp_keyword_for_search);
164 }

```

## 5. Sort

File	Function
<b>manager_mode.c</b>	manager_mode
<b>database.c</b>	cp_output num_ques insert_in_arr

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

```
112         else if(opr==4){
113             // "4. Display all questions in descending order\n"
114             cp_output();
115         }
```

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

```
348 void cp_output(){
349     int question_num=num_ques(root_qid);
350     struct node* arr[question_num];
351     insert_in_arr(arr,root_qid);
352     for(int i=0;i<question_num;i++){
353         for( int j=0;j<question_num-i-1;j++){
354             if(arr[j]->que.correct_percent > arr[j+1]->que.correct_percent){
355                 struct node* temp=arr[j];
356                 arr[j]=arr[j+1];
357                 arr[j+1]=temp;
358             }
359         }
360     }
361     for(int i=0;i<question_num;i++){
362         printf("correct rate:%.2f,content:%s\n",arr[i]->que.correct_percent*100,arr[i]->que.q_content);
363     }
364 }
```

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

```
288 int num_ques(struct node* root){
289     if(NULL == root) return 0;
290     return 1+num_ques(root->lchild_qid)+num_ques(root->rchild_qid);
291 }
```

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.

```
366 void insert_in_arr(struct node** arr,struct node* root){
367     if(NULL == root)return;
368     static int counter=0;
369     arr[counter]=root;
370     counter++;
371     insert_in_arr(arr,root->lchild_qid);
372     insert_in_arr(arr,root->rchild_qid);
373 }
```

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

## 6. 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.

```
12      //database input from file
13      FILE *fptr;
14      fptr = fopen("test.txt", "r");
15      if (fptr == NULL) {
16          printf("fail to open!\n");
17          return;
18      }
```

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

```
66      for (int i = 1; i <= max_id(); i++) { //there some bug that will fprintf id = 0
67          struct ques *temp_ques = search_ID_ques(i);
68          if(temp_ques == NULL){
69              continue;
70          }
71          fputs(temp_ques->q_content, fptr_out);
72          fprintf(fptr_out, "\n%d\n", temp_ques->ans);
73
74          //renew answered_num and correct_num
75          fprintf(fptr_out, "%.2f\n", temp_ques->correct_percent);
76          fprintf(fptr_out, "%d\n", temp_ques->answered_num);
77          fprintf(fptr_out, "%d\n", temp_ques->correct_num);
78      }
```

# 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.

```
211 struct node* balance_qid(struct node* root){
212     update_qid(root);
213     int balanceFactor = getBalanceFactor_qid(root);
214     if(balanceFactor > 1){
215         if(getBalanceFactor_qid(root->lchild_qid) < 0){
216             root->lchild_qid = rotateleft_qid(root->lchild_qid);
217         }
218         return rotateright_qid(root);
219     }
220     if(balanceFactor < -1){
221         if(getBalanceFactor_qid(root->rchild_qid) > 0){
222             root->rchild_qid=rotateright_qid(root->rchild_qid);
223         }
224         return rotateleft_qid(root);
225     }
226     return root;
227 }
```



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

```
229     int getheight_qid(struct node* node){
230         if(NULL == node){
231             return 0;
232         }
233         else return node->height_qid;
234     }
```

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

```
236     int getBalanceFactor_qid(struct node* node){
237         if(NULL == node) return 0;
238         else return getheight_qid(node->lchild_qid)-getheight_qid(node->rchild_qid);
239
240     }
```

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

```
242     struct node* rotateright_qid(struct node* node){
243         struct node* newroot=node->lchild_qid;
244         node->lchild_qid = newroot->rchild_qid;
245         newroot->rchild_qid=node;
246         update_qid(node);
247         update_qid(newroot);
248         return newroot;
249
250     }
```

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

```
252     struct node* rotateleft_qid(struct node* node){
253         struct node* newroot=node->rchild_qid;
254         node->rchild_qid=newroot->lchild_qid;
255         newroot->lchild_qid=node;
256         update_qid(node);
257         update_qid(newroot);
258
259         return newroot;
260     }
```

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

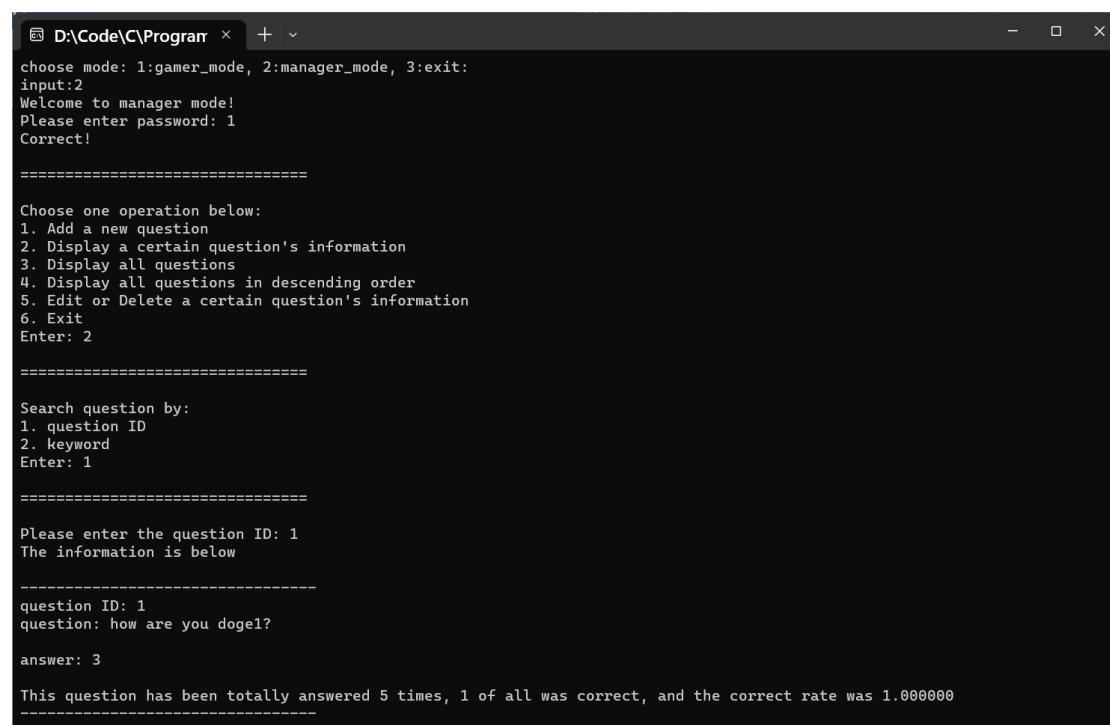
Which is detailed described at p.10~12.

## 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:



```
D:\Code\C\Program x + -
choose mode: 1:gamer_mode, 2:manager_mode, 3:exit:
input:2
Welcome to manager mode!
Please enter password: 1
Correct!

=====

Choose one operation below:
1. Add a new question
2. Display a certain question's information
3. Display all questions
4. Display all questions in descending order
5. Edit or Delete a certain question's information
6. Exit
Enter: 2

=====

Search question by:
1. question ID
2. keyword
Enter: 1

=====

Please enter the question ID: 1
The information is below

-----
question ID: 1
question: how are you doge1?
answer: 3

This question has been totally answered 5 times, 1 of all was correct, and the correct rate was 1.000000
=====
```

## 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.

**Wow! You answered 10 questions in a row!**

## Another design: Manager Password

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

```
50      // identification
51      char password[100]; // but less than 21
52      printf("Please enter password: ");
53      scanf("%s", password); // input until \n
54
55      if(strcmp(password, "1"/* the correct password is set here */ ) == 0){
56          printf("Correct!\n");
57      }
58      else{
59          printf("Error!\n");
60          return;
61      }
```

## Demonstration

The link to the **GitHub repository**:

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