Study Guide Project Improvement

Tong Yi

In this project, we display questions and answer them. If the answer is correct, we display true, otherwise, we display false. This project can serve as a study guide for our courses or Civics (History and Government) Questions for the Naturalization Test, as in `https://www.uscis.gov/sites/default/files/document/questions-and-answers/100q.pdf`.

Warning:

1. These are copyrighted materials and cannot be uploaded to the Internet.
2. Only ask help from teaching staff of this course.
3. Use solutions from ChatGPT or online tutoring websites like, but not limited to, chegg.com, violates academic integrity and is not allowed.

1 Task A: organize the program into functions

Motivation: In previous tasks, we put every code in main function. Also, we enter a file name and read its contents. We may need to read several files, for example,
`cs135_midterm_f24_v1.txt` and `cs135_midterm_f24_v2.txt`.
Then add the questions to an array of Questions.
In Task A, we do the following.

1. Name your source code `checkAnswer_function.cpp`.
2. Define function

   `void read_file(string fileName, Question ques[], int capacity, int& size);`

   Note that the return type is void, but the size of the array may be increased after reading a file, so size needs to be passed by reference. See the & near type `int` for parameter size?

   (a) Given struct Question as follows.

```
struct Question {
    string text; //question text
    string answer;
    string explanation;
    string version;
    string type;
    string label;
};
```

(b) Read a file whose name is saved in `fileName`.

(c) Read each question in the file, if the current size Question array `ques` does not equal to the `capacity` of the array, add to the end of the array. Increase the current size by 1.

3. Download **link to cs135 midterm f24 v1**.

4. Download **link to cs135 midterm f24 v2**.

5. Download **link to cs135 midterm s24 v1**.

6. Define `void display(Question ques[], int size)` function, which displays the fields of each question in array `ques`. You need to fill in the ... parts.

```
void display(Question ques[], int size) {
    for (int i = ...; i < ...; ...) {
        cout << i + 1 << endl; //start labeling from 1

        //display question text of the ith question
        cout << "question: " << ... << endl;

        //display answer of the ith question
        cout << "answer: " << ... << endl;

        //display explanation of the ith question
        cout << "explanation: " << ... << endl;
        cout << "type: " << ... <<  endl;
        cout << "version: " << ... <<  endl;
        cout << "label: " << ... <<  endl;
        cout << endl;
    }
}
```

7. In main function, test `read_file` function as follows.

```
int main() {
    const int CAPACITY = 1000;
    Question ques[CAPACITY]; //question array

    int size = 0;

    //TODO: call read_file for "cs135_midterm_f24_v1.txt",
    //save the questions in array ques if the capacity is not yet reached.

    //TODO: call read_file for "cs135_midterm_f24_v2.txt",
    //save the questions in array ques if the capacity is not yet reached.

    //TODO: call read_file to read "cs135_midterm_s24_v1.txt"

```

```
    //TODO: call display function on array ques.
    //Do not forget to pass the size of array ques,
    //that is, the number of elements in ques,
    //as the second parameter.

    return 0;
}
```

Here is a sample output (the number of empty lines between fields might be a little different).

```
1
question: Given char arr[] = {'A', 'B', 'C'}, what is arr[1]?
answer: 'B'
explanation: arr[1] is the second element of array arr, which is 'B' in this
   example.

type: array
version: f24 v1
label: 1.1

2
question: Declare function increase, given an integer array arr with size many
   elements, increase each element of the array by 1. Return type is void. Define
   the function header (no implementation is needed).
answer: void increase(int arr[], int size);
explanation: (1) the first parameter is int arr[], the name of array arr, which
   also implies the address of the first element of array.
(2) the second parameter represents the number of elements of the array.

type: function; array
version: f24 v1
label: 1.2

3
question: Assume that n is properly declared and initialized. Write a statement to
   declare lastDigit as an integer and initialize it to be the least significant
   digit of integer n. Suppose n is 123, after the statement, lastDigit is 3.
answer: int lastDigit = n % 10;
explanation: (1) operator % is called remainder or modular operator.
(2) For example, 12 % 10 means the remainder when dividing 12 pens among 10
   students, each student gets 1 pen, and there are 2 pens left.
(3) In general, n % 10 returns the last digit, or the rightmost digit (least
   significant digit), of n.
(4) int lastDigit = n % 10; is a statement to declare lastDigit as an int and
   initialize it by the last digit of n.

type: arithmetic; modular; remainder
```

```
version: f24 v1
label: 1.3

... //omit the contents

29
question: What is the output for the following code?
#include <iostream>
using namespace std;

void foo(int& a, int b);

int main() {
    int i = 1;
    int j = 3;
    foo(i, j);
    cout << "i = " << i
         << ", j = " << j << endl;

    return 0;
}

void foo(int& a, int b) {
    a++;
    b--;
}


answer: i = 2, j = 3
explanation:
type: function; pass by value; pass by reference
version: s24 v1
label: 1.9

30
question: Write a condition to represent that char variable ch is none of the
   following: 'a', 'b', or 'c'.
answer: (ch != 'a' && ch != 'b' && ch != 'c')
explanation: another solution is (! (ch == 'a' || ch == 'b' || ch == 'c'))

type: condition
version: s24 v1
label: 1.10
```

2 Task B: more functions

The goal is to organize the tasks into functions, each concentrate on a task. In this way, we can unit test a function before moving to the next.

Here is an outline.

1. Name your source code `checkAnswer_function_rand.cpp`.
2. Extract types from `type` field of `Question`, which is separated by semicolons (symbol ;).
3. The type field of a question may involve more than one type. As a result, we need to use an array to hold each type.
   (a) If a type is
   `integer division; arithmetic`
   Then the returned array has two elements: `integer division` and `arithmetic`.
   (b) If a type is
   `integer division; arithmetic; remainder`
   Then the returned array has three elements: `integer division`, `arithmetic` and `remainder`.
4. Then add the types to an array of strings. Make sure that the array is sorted.
5. Define a function to answer questions based on types.
6. Define a function to give feedback.

In this task, we define functions, let each function concentrate on a task, then main function calls the key functions.

2.1 Define function trim

Define a function to trim (remove) all the spaces before the leftmost non-space character and all the spaces after the rightmost non-space character from a string.

```
1 string trim(string str);
```

For example, `trim("    hello, world  ")` should return `"hello, world"`.

1. Why do we need the function? When we extract possible types from `type` field of a question, the extracted result might contain spaces before the first non-space character and the after the last non-space character, to avoid adding `" array "` and `"array"` to the array at the same time, we only add a type without those extra spaces.
2. Here are some hints for this function.
   (a) Find out the index of first non-space character of a string using `find_first_not_of` method. Or, search the string from the first character to the last character until there is no more character in the string or the first non-space character is encountered. Save that index.

(b) Find the index of the last non-space character of a string using `find_last_not_of` method. Or, search the string **backwards** from the last character to the first one until there is no more character or a non-space character is found. Save the index.

(c) Use `substr` method to find out the substring between those two indices (inclusive). Note that the first parameter of `substr` method in C++ is the index of the first character of the resulting substring in the original string, and the second parameter is the number of characters in the substring.

2.2 Define function count_occurrence

Define a function to count number of occurrences of a char `ch` in string `str`.

```
1 int count_occurrence(string str, char ch);
```

1. Refer to Lab 7A for help. This function belongs a category called counting. Given an array, find out the number of elements of a feature (equal to something, longer than or larger than something, and so on).

2. A string can be thought as an array of characters (type char) with methods to find out number of elements using its `length` or `size` method.

   Warning: an array of C++ is not a class, so dot operator does not apply. Specifically, the size of an array needs to be given explicitly in a function.

   In general, codes for array can be modified to work on similar problems for strings.

3. Call the above function to find out the number of semicolons of the type field of a question to find out the number of elements in a dynamically allocated array in function `extract_type`.

   If the number of semicolons is n, then the number of types in this type field is n+1. For example, if there are two semicolons like the type field `integer division; arithmetic; remainder`, then the number of individual types is 3.

   We need this number as the size to apply for a dynamically allocated array of strings to hold the types in that type field.

2.3 Define function extract_type

Extract types separated by semicolon symbol (;) and put the types in an array of dynamically allocated string array.

```
1 string* extract_type(string type, int& num_types_curr_item);
```

The return type is `string*`, which saves the initial address of the dynamically allocated memory.

1. Need a parameter to hold the number of elements in this array. Since a function in C++ can return at most one value, we need to work around this limit by pass a parameter by reference as shown in `int& num_types_curr_item`, where `&` after `int` means the following parameter pass by reference.

   Pass by reference means the formal parameter is the **original** actual parameter (aka argument). Whatever change made on the formal parameter in the function is carried back to its caller.

By contrast, pass by value – no & after the type in formal parameters in function header – means to a formal parameter is the **duplicated** copy of the actual parameter. Whatever changes made on the duplicated copy, it will not affect the actual parameter in the caller function, unless the value is returned.

Note that if a formal parameter is an array, which implies the address of the first element of the array, then the contents of the array can be modified. Reason: each element has a unique address. Once the address is know, we can access and may change the elements.

2. Here are some hints.

   (a) Call `count_occurrences` to find out number of semicolon symbol (;) in `type`, add 1 to this value (why?) and save it to `num_types_curr_item`.

   (b) Apply a dynamically allocated array of strings with size `num_types_curr_item`.

   (c) Extract the corresponding types in `type` separated by semicolon (;).

   (d) Trim those obtained types and save them to the dynamically allocated array.

   (e) Return the dynamically allocated array. No need to sort the array.

      i. For example, if the argument for `type` is `integer division; array; remainder`, apply a dynamically allocated array of size 3. Initialize with elements `"integer division"`, `"array"`, and `"remainder"`. Return the dynamically allocated array.

3. Students may wonder what the difference of statically allocated memory and dynamically allocated memory are.

   (a) Statically allocated array memory, whose size is a const int, is allocated in the stack or a separate data segment (also known as the static memory region).

   (b) Dynamically allocated array memory, whose size can be an int variable, is allocated from heap.

   (c) Statically allocated memory is released once a function finishes and returns to its caller. So a statically allocated array declared and initialized in a callee function cannot return to its caller.

   (d) Dynamically allocated array applied in a callee function can be carried back to its caller. That is, dynamically allocated memory allocated in a callee function can be kept even after the callee function finishes.

   (e) Related example: you may rent a textbook for 120 days. After the expiration date, the book is returned to the publisher, whether you like it or not. The publisher removes the book from your bookshelf.

   This is like statically allocated memory is released back to the operating system automatically when the function ends.

   On the other hand, my textbook came from a different branch (memory) of the publisher (system). My book is like dynamically allocated memory. I can keep it until I no longer need the book and then return it to the publisher. The return is initialized from me.

   It is like I use `new` (fill an application) to apply for an instructor book from the publisher. When I no longer need the book (memory), I use `delete` (remove the book from my bookshelf) to return it.

2.4 Define function insert_order_unique

```
1 void insert_order_unique(string types[], int type_capacity, int& type_count, string
    toAdd);
```

These are parameters.

1. `types`: an `already-sorted` array of strings
2. `type_capacity`: capacity of array `types`, that is, the maximum number of elements array `types` can hold.
3. `type_count`: number of current elements in `types`
4. `toAdd`: a string to be added to `types`
5. Goal: insert `toAdd` to `types` if there the string does not appear in the array and the array is not full. The array is sorted before insertion and need to be sorted after insertion. Here are some hints.
   (a) If the array is full, that is, `type_count` is larger than or equal to `type_capacity`, do nothing and return.
   (b) Traverse through the array until the end is reached or the elements are smaller than `toAdd`. String a is smaller than string b means a appears in a dictionary before b. For example, `"success"` is smaller than `"work"` since `"success"` appears before `"work"` in dictionary.
   (c) If the end is reached, that means all current elements are smaller than `toAdd`, add `toAdd` to the end.
   (d) Otherwise, there is an element in the array is larger than or equal to `toAdd`.
      i. If they are equal, no need to add a type already saved in the array, do nothing and return to the caller.
      ii. Otherwise, move the element in the array and all its right side neighbor one spot to the right to make room for `toAdd`.
      iii. Put `toAdd` to the correct position.
      iv. After `toAdd` is inserted, increase `type_count` by 1.
6. Here is a summary on the sorting algorithms we have learned so far.
   (a) Learn bubble sort on 3/6/25. Key idea: swap out-of-order adjacent pairs from beginning to the end, doing so would put the largest element to the rightmost position.
      i. Repeat the above process to put the second largest element to the second-to-rightmost position. Keep the process until $n-1$ out of n elements are in their correct position after sorting, where n is the number of element of the array.
      ii. In Problem 1.9 of midterm of Spring 25, we test bubble sort in descending order.
   (b) On 3/10/25, we share ideas of homework E5.14 and E5.15, to put four elements `a, b, c`, and `d` in order.
      i. First, if `a` is larger than `b`, we swap `a` and `b`. Now `a` is the smaller of `a` and `b`.

ii. Second, if `a` is larger than `c`, swap `a` and `c`. Now `a` is the smallest of `a, b`, and `c`.

iii. Third, if `a` is larger than `d`, swap `a` and `d`. Now `a` is the smallest of `a, b, c,` and `d`.

iv. Repeat the above process to work with `b, c,` and `d`.

v. In general, select the smallest element of an array and put it in the first position of the array. Select the second smallest element of the array and put it in the second position of the array, and so on. This is the key idea of selection sort.

(c) In this project, we learn insertion sort, insert an element to a sorted array and keep it sorted after insertion.

(d) Bubble sort, selection sort, and insert sort are not efficient. The most efficient sorting algorithm is quick sort.

2.5 Define function insert_order_unique for types of each question

```
1 void insert_order_unique(string types[], int type_capacity, int& type_count, Question
    ques[], int ques_size);
```

1. You may notice the function has exact the same name as the above function but the parameter list is different. This is called `function overload`. That is, two or more functions have

   (a) exactly the same name

   (b) similar functionality

   (c) different parameter lists. The difference can be shown in

      i. number of parameters
      For example, substr(int) and substr(int, int). The first function takes only one parameter, while the second one takes two parameters.

      ii. type of parameters
      For example, `int foo(int a)` and `string foo(string str)`, where `foo` is a function name when we do not want to delve in details. It like we use John Doe to represent a person.

      iii. order of parameters
      For example, `int foo(int, string)` is different from `int foo(string, int)`.

   (d) It does not matter whether the return types are the same or not.

   (e) An example in daily life involves two washing machines.

      i. One is used in home, which has only an input slot for laundries. It is similar to a function with only one input parameter.

      ii. The other is for commercial usage, which has one more input slot to take in money. It is similar to a function with two input parameters.

      iii. Both machines provide similar functionalities called wash, no need to use two different names `home_wash` and `commerical_wash`.

2. In this function, for each question,

(a) Call the following function to extract each type from `type` field of a question.

```
string* extract_type(string type, int& num_types_curr_item);
```

(b) Call the following version to insert each type to array `types`.

```
void insert_order_unique(string types[], int type_capacity, int& type_count,
    string
    toAdd);
```

(c) Do not forget to release dynamically allocated memory returned from `extract_type` function. Afterwards, `types` save all types, once and exactly once, from the type field of all questions in sorted order.

2.6 Define function `choose_type`

The function header shows as follows.

```
string choose_type(string* types, int type_count);
```

The goal of this function is to list available types from sorted array `types`, where the label of the first type starts from 1. If 0 is chosen, then all types are chosen, which is represented by an empty string.

Do the following.

1. Print string `"0. ALL TYPES"` with a new line.
2. Print items from sorted array `types`, where the label of the first type starts from label 1. The label of the last type is `type_count`.
3. Ensure to choose an integer in `[0, type_count]`, where both ends are included. If the number is out of range, prompt the user to select again.
4. If a user chooses 0, then return an empty string reprsenting all types, otherwise, return the corresponding item in array `types`. Note that the label starts from 1, but the index starts from 0, so there is an offset by 1.

2.7 Define function `randomize`

Sometimes we work on the problems in a fixed order and memorize the answer. To avoid this situation, we randomize array of questions.

```
void randomize(Question ques[], int size);
```

You **must** follow the ideas listed below, or your code will not pass gradescope.

1. Declare an integer variable to save number of elements to be randomized. Call it `numToRandomize`. Initialize it to be `size`.
2. Choose a random integer in `[0, numToRandomize)`. That is, a random integer that is at least 0 but is smaller than `numToRandomize`. That is a valid index of the sub-array of the first `numToRandomize` elements.

3. The above step is like to choose the element indexed at that generated random number. To avoid that element to be chosen again, swap the element with the element indexed at `numToRandomize -1`, the last element in the sub-array with the first `numToRandomize` elements.

4. Reduce `numToRandomize` by 1.

5. Repeat the above process by going back to Step 2 until `numToRandomize` is reduced to 1.

2.8 Define function `feedback`

Based on the number of problems correctly answered and the total number of problems worked, print feedback to users.

```
void feedback(int numCorrect, int numQuestions);
```

Do the following in the function.

1. Print out the number of correctly answered questions, saved in parameter `numCorrect` in a line.

2. Calculate and and print the percentage based on `numCorrect` and `numQuestions`. The latter is the total number of questions worked.

3. If percentage is at least 90%, print `"excellent"`, otherwise, if percentage is at least 80%, print `"good"`, otherwise, if percentage is at least 70%, print `"pass"`, otherwise, print `"please ask help ASAP"`.

2.9 Define function answer_by_type

```
void answer_by_type(Question ques[], int size, string chosenType);
```

Given array `ques` of questions and its `size`, select all questions whose type contains `chosenType`. That is, search `chosenType` from the type field of each element of `ques`. Each user has at most 3 tries to answer a question. After providing three wrong answers, or, correctly answer the question in at most three tries, move to the next question in array of Questions. Record the number of correctly answered questions.

Call `feedback` function and provide feedback to users.

2.10 When to call cin.ignore(INT_MAX, ' n') statement?

The above statement ignores the remaining characters, including new line character, left in the keyboard buffer.

The above line is useful in the following pseudocode.

```
//File name: use_cin_ignore.cpp
//sample input:
//Enter your age: 18
//Enter your major: Your age is 18
//Your major is

#include <iostream>
```

```cpp
#include <string>
using namespace std;

int main() {
    cout << "Enter your age: ";
    int age;
    cin >> age;

    cout << "Enter your major: ";
    string major;
    getline(cin, major);
      //major might contain spaces,
      //so we use getline function to read the whole line.

    cout << "Your age is " << age << endl;
    cout << "Your major is " << major << endl;
    return 0;
}
```

When `getline(cin, stringVariable);` follows`cin >> variable;`.

1. To finish input, need to enter return key, which is `\n` new line character.
2. However, Operator `>>` only takes input before `\n`.
3. If statement `getline(cin, stringVariable);` follows, then getline statement takes everything before and at `\n`. So it is like to `stringVariable` is set to be an empty string. This might not be what we want.
4. Fix: add `cin.ignore(INT_MAX, '\n');` to clear the keyboard buffer.

```cpp
//sample input / output:
//Enter your age: 18
//Enter your major: computer science
//Your age is 18
//Your major is computer science

#include <iostream>
#include <string>
using namespace std;

int main() {
    cout << "Enter your age: ";
    int age;
    cin >> age;

    cin.ignore(INT_MAX, '\n'); //ADD
      //ignore the remaining contents of keyboard buffer.
```

```
18
19      cout << "Enter your major: ";
20      string major;
21      getline(cin, major); //major might contain spaces
22      cout << "Your age is " << age << endl;
23      cout << "Your major is " << major << endl;
24      return 0;
25  }
```

2.11 Unit test each function

Here is `unit_test.cpp`, downloaded from **link to unit_test.cpp**, which tests each function. Note that the following `main` function takes parameters, parameter `argv` is the number of parameters, parameter `argc` is an array of char* – an array of chars, string variable in C – to hold those parameters.

To test functions of `checkAnswer_function_rand.cpp`, do the following.

1. We have two files, `checkAnswer_function_rand.cpp` and `unit_test.cpp`.
2. File `checkAnswer_function_rand.cpp` is source code of our project.
3. Both files have main functions. However, each C++ project can have only one main function.
4. To use the main function from `unit_test.cpp`, change the name of function main of `checkAnswer_function_` to be main2 when we run unit test for functions in the above source code.

   In terminal, run the following command with return key.

   (a) `g++ -std=c++20 unit_test.cpp -o test`

   If there is no error, a runnable file called test is generated. Run the following commands to test functions of our source code.

   ./test 3

   ./test 4

   ...

   ./test c

5. Remember to change function main2 back to main when running `checkAnswer_function_rand.cpp`.

```
1  #include <iostream>
2  #include <string>
3  #include <climits> //INT_MAX
4  #include "checkAnswer_function_rand.cpp"
5
6  //How to run this file:
7  //1. We have two files, checkAnswer_function_rand.cpp and unit_test.cpp.
8  //2. File checkAnswer_function_rand.cpp is the source code of our project.
9  //3. unit_test.cpp is to test functions defined in checkAnswer_function_rand.cpp.
```

```
//4. Both files have main functions. However, each C++ project can have only one main function.
//5. To run the main function in unit_test.cpp, change the name of function main of checkAnswer_function_rand.cpp to be main2.
//6. g++ -std=c++20 unit_test.cpp -o test
//7. If there is no error in unit_test.cpp, a runnable file called test is generated, run the following commands with return key to test each function.

//./test 1
//or
//./test 2
//./test 3
//...
//./test c

//8. Change main2 function in checkAnswer_function_rand.cpp back to main.

void read_file_into_array(Question ques[], int capacity);

int main(int argc, char** argv) {
    if (argc < 2) {
       std::cout << "missing argument in main function" << std::endl;
       exit(0);
    }

    //test answer_by_type
    Question q1;
    q1.text = "Given char arr[] = {'A', 'B', 'C'}, what is arr[1]?";
    q1.answer = "'B'";
            q1.explanation = "arr[1] is the second element of array arr, which is 'B' in this example.";
            q1.version = "f24 v1";
            q1.label = "1.1";
            q1.type = "array";

            Question q2;
            q2.text = "Declare function increase, given an integer array arr with size many elements, increase each element of the array by 1. Return type is void. Define the function header (no implementation is needed).";
            q2.answer = "void increase(int arr[], int size);";
            q2.explanation = "(1) the first parameter is int arr[], the name of array arr, which also implies the address of the first element of array.\n(2) the second parameter represents the number of elements of the array.";
            q2.version = "f24 v1";
            q2.label = "1.2";
            q2.type = "function; array";
```

```
48
49          Question q3;
50          q3.text = "Assume that n is properly declared and initialized. Write a
    statement to declare lastDigit as an integer and initialize it to be the least
    significant digit of integer n. Suppose n is 123, after the statement, lastDigit is
    3.";
51          q3.answer = "int lastDigit = n % 10;";
52          q3.explanation = "(1) operator % is called remainder or modular operator.\n
    (2) For example, 12 % 10 means the remainder when dividing 12 pens among 10 students
    , each student gets 1 pen, and there are 2 pens left.\n(3) In general, n % 10
    returns the last digit, or the rightmost digit (least significant digit), of n.\n(4)
     int lastDigit = n % 10; is a statement to declare lastDigit as an int and
    initialize it by the last digit of n.";
53          q3.version = "f24 v1";
54          q3.label = "1.3";
55          q3.type = "arithmetic; modular; remainder";
56
57          Question q4;
58          q4.text = "What is the output?\n\nstring tens_name(int n);\n\nint main() {\
    n    cout << tens_name(82) << endl;\n    return 0;\n}\nstring tens_name(int n) {\n
      if (n < 20 || n > 99)\n        return \"\";\n\n    string names[] = {\"\", \"\",
    \"twenty\", \"thirty\", \"forty\", \"fifty\", \"sixty\", \"seventy\", \"eighty\", \"
    ninety\"};\n    return names[n / 10];\n}";
59          q4.answer = "eighty";
60          q4.explanation = "(1) When calling tens_name(82), n in tens_name is
    initialized to be 82.\n(2) Since 82 is not less than 20 or 82 is not larger than 99,
     no return \"\";\n(3) 82 / 10 is integer division. It is like to divide 82 pens
    among 10 students, each student get 8 pens. So 82 / 10 returns 8.\n(4) names[n / 10]
     is names[82 / 10], which is names[8].\n\nindex    0  1      2       3        4
      5        6       7      8\n\nelement
    +--+--+--------+--------+-------+-------+-------+---------+--------+...\n
    |\"\"|\"\"|\"twenty\"|\"thirty\"|\"forty\"|\"fifty\"|\"sixty\"|\"seventy\"|\"eighty
    \"|...\n +--+--+--------+--------+-------+-------+-------+---------+--------+...\n\n
    (5) The return of tens_name(82) is \"eighty\".\n(6) In main function, print
    tens_name(82), so the print out is \"eighty\" (without quotes).";
61          q4.version = "f24 v1";
62          q4.label = "1.4";
63          q4.type = "integer division; array";
64
65          Question ques[] = {q1, q2, q3, q4};
66
67          int size = sizeof(ques) / sizeof(ques[0]);
68
69      switch (*argv[1]) {
70          case '1': {
71              const int CAPACITY = 1000;
```

```
72              Question ques[CAPACITY]; //question array
73
74              read_file_into_array(ques, CAPACITY);
75              break;
76          }
77
78          case '2': {
79              //test when capacity is reached
80              const int CAPACITY = 25;
81              Question ques[CAPACITY]; //question array
82
83              read_file_into_array(ques, CAPACITY);
84              break;
85          }
86
87          case '3': {
88              //test trim
89              //the string is given randomly in gradescript
90              cout << "Enter a string: ";
91              string inputStr;
92              getline(cin, inputStr);
93              //cout << "input String: " << inputStr << endl;
94
95              string trimmedStr = trim(inputStr);
96              cout << "\""
97                   << trimmedStr << "\"";
98              break;
99
100 //Run ./test  3
101 //sample input / output
102 //Enter a string:    Hello, World
103 //"Hello, World"
104         }
105         case '4': {
106             //test string* extract_type(string type, int& num_types_curr_item)
107             string types[] = {"function; array",
108                "arithmetic; modular; remainder",              "integer division; array
    " };
109             for (string type : types) {
110                 int num_types_curr_item = 0;
111                 string* pStr = extract_type(type, num_types_curr_item);
112                 for (int i = 0; i < num_types_curr_item; i++)
113                     cout << pStr[i] << endl;
114
115                 //release dynamically allocated memory
116                 delete[] pStr;
```

```
117                 pStr = nullptr;
118             }
119 
120             break;
121 
122 //run
123 //./test 4
124 //Sample input/output:
125 //function
126 //array
127 //arithmetic
128 //modular
129 //remainder
130 //integer division
131 //array
132         }
133 
134         case '5': {
135             //test int count_occurrences(string str, char ch)
136             string strs[] = {"integer division; array", "arithmetic; modular; remainder
    "};
137             int size = sizeof(strs) / sizeof(strs[0]);
138             for (int i = 0; i < size; i++) {
139                 cout << count_occurrences(strs[i], ';') << endl;
140                 cout << count_occurrences(strs[i], 'a') << endl;
141             }
142 
143             break;
144 //Run ./test 5 in terminal
145 //Here is the sample input / output:
146             //1
147             //2
148             //2
149             //3
150         }
151 
152         case '6': {
153         //insert_order_unique(string types[], int type_capacity, int& type_count,
    string toAdd)
154              const int TYPE_CAPACITY = 30;
155              string types[TYPE_CAPACITY];
156              int type_count = 0;
157 
158              string elmsToAdd[] = {"function",
159                  "array",
160                  "integer division",
```

```
                "array"
            };

            for (string str: elmsToAdd) {
                insert_order_unique(types,                TYPE_CAPACITY, type_count,
    str);

                cout << type_count << endl;
                for (int i = 0; i < type_count;
                    i++) {
                    cout << types[i] << endl;
                }
            }
            break;

//Run ./test 6 with return key in terminal.
//Sample output:
            //1
            //function
            //2
            //array
            //function
            //3
            //array
            //function
            //integer division
            //3
            //array
            //function
            //integer division
        }

//void insert_order_unique(string types[], int type_capacity, int& type_count, Question
    ques[], int ques_size);
        case '7': {
            const int QUES_CAPACITY = 1000;
            Question ques[QUES_CAPACITY];
            int ques_size = 10;
            ques[0].type = "array";
            ques[1].type = "function; array";
            ques[2].type = "arithmetic; modular; remainder";
            ques[3].type = "integer division; array";
            ques[4].type = "string; substring";
            ques[5].type = "arithmetic; integer division";
            ques[6].type = "arithmetic; integer division";
            ques[7].type = "repetition";
```

```
205             ques[8].type = "function";
206             ques[9].type = "condition";
207 
208             const int TYPE_CAPACITY = 30;
209             string types[TYPE_CAPACITY];
210             int type_count = 0;
211             insert_order_unique(types, TYPE_CAPACITY, type_count, ques, ques_size);
212 
213             for (int i = 0; i < type_count; i++)
214                 cout << i + 1 << ". " << types[i] << endl;
215 
216            break;
217 
218 //Run ./test 7 with return key in terminal,
219 //here is a sample output.
220 //1. arithmetic
221 //2. array
222 //3. condition
223 //4. function
224 //5. integer division
225 //6. modular
226 //7. remainder
227 //8. repetition
228 //9. string
229 //10. substring
230         }
231 
232         case '8': {
233             const int SIZE = 10;
234             Question ques[SIZE];
235             for (int i = 0; i < SIZE; i++) {
236                 string str = "1.";
237                 ques[i].label = str + to_string(i+1);
238             }
239 
240             randomize(ques, SIZE);
241 
242             //Run in linux environment like onlinegdb to get same running out in
    gradescope.
243             for (int i = 0; i < SIZE; i++)
244                 cout << ques[i].label << endl;
245 
246             break;
247 //output in Mac when no srand setting up
248 //Run ./test 8 with return key in terminal
249 //1.4
```

```
//1.9
//1.6
//1.1
//1.3
//1.5
//1.7
//1.2
//1.10
//1.8
       }
        case '9': {
             answer_by_type(ques, size, "array");
             break;

//Run the following command with return key in terminal
//./test 9
//sample input/output:
//question f24 v1 1.1: Given char arr[] = {'A', 'B', 'C'}, what is arr[1]?
//Enter you answer: 'B'
//number of tries: 1
//true
//
//question f24 v1 1.2: Declare function increase, given an integer array arr with size
   many elements, increase each element of the array by 1. Return type is void. Define
   the function header (no implementation is needed).
//Enter you answer: void increase(int arr[], int size);
//number of tries: 1
//true
//
//question f24 v1 1.4: What is the output?
//
//string tens_name(int n);
//
//int main() {
//    cout << tens_name(82) << endl;
//    return 0;
//}
//string tens_name(int n) {
//    if (n < 20 || n > 99)
//        return "";
//
//    string names[] = {"", "", "twenty", "thirty", "forty", "fifty", "sixty", "seventy
   ", "eighty", "ninety"};
//    return names[n / 10];
//}
//Enter you answer: eighty
```

```
//number of tries: 1
//true
//
//number of correct problems: 3
//percentage of correct: 100%
//excellent
        }

        case 'a': {
             answer_by_type(ques, size, "");
             break;

//Run the following command with return key in terminal
//./test a
//sample input/output:
//question f24 v1 1.1: Given char arr[] = {'A', 'B', 'C'}, what is arr[1]?
//Enter you answer: 'a'
//number of tries: 1
//false
//Enter you answer: 'A'
//number of tries: 2
//false
//Enter you answer: 'B'
//number of tries: 3
//true
//
//question f24 v1 1.3: Assume that n is properly declared and initialized. Write a
   statement to declare lastDigit as an integer and initialize it to be the least
   significant digit of integer n. Suppose n is 123, after the statement, lastDigit is
   3.
//Enter you answer: int lastDigit = n % 10;
//number of tries: 1
//true
//
//question f24 v1 1.2: Declare function increase, given an integer array arr with size
   many elements, increase each element of the array by 1. Return type is void. Define
   the function header (no implementation is needed).
//Enter you answer: void increase(int arr[], int size);
//number of tries: 1
//true
//
//question f24 v1 1.4: What is the output?
//
//string tens_name(int n);
//
//int main() {
```

```
//    cout << tens_name(82) << endl;
//    return 0;
//}
//string tens_name(int n) {
//    if (n < 20 || n > 99)
//        return "";
//
//    string names[] = {"", "", "twenty", "thirty", "forty", "fifty", "sixty", "seventy
    ", "eighty", "ninety"};
//    return names[n / 10];
//}
//Enter you answer: eighty
//number of tries: 1
//true
//
//number of correct problems: 4
//percentage of correct: 100%
//excellent
        }

        case 'b': {
            string types[] = {
              "arithmetic",
              "array",
              "condition",
              "function",
              "integer division",
              "modular",
            };
            int size = sizeof(types) / sizeof(types[0]);

            string chosenType = choose_type(types, size);
            cout << "chosen type: \"" << chosenType << "\"" << endl;

            cout << "\n\nTest when users choose 0. ALL TYPES\n\n";
            //run choose_type the second time
            chosenType = choose_type(types, size);
            cout << "chosen type: \"" << chosenType << "\"" << endl;
            break;

//Run the following command with return key in terminal
//./test b
//sample input/ouput:
//0. ALL TYPES
//1. arithmetic
//2. array
```

```
//3. condition
//4. function
//5. integer division
//6. modular
//Enter a type: 1
//chosen type: "arithmetic"
//
//
//Test when users choose 0. ALL TYPES
//
//0. ALL TYPES
//1. arithmetic
//2. array
//3. condition
//4. function
//5. integer division
//6. modular
//Enter a type: 0
//chosen type: ""
        }

        case 'c': {
           //void feedback(int numCorrect, int numQuestions)
            int numQuestions = 7;
            for (int numCorrect = 4;
                 numCorrect <= numQuestions;
                 numCorrect++)
                feedback(numCorrect, numQuestions);

            break;

//Run the following command with return key in terminal
//./test c

//sample input/output
//number of correct problems: 4
//percentage of correct: 57.1429%
//please ask help ASAP
//number of correct problems: 5
//percentage of correct: 71.4286%
//pass
//number of correct problems: 6
//percentage of correct: 85.7143%
//good
//number of correct problems: 7
//percentage of correct: 100%
```

```
//excellent
        }
    }

    return 0;
}

void read_file_into_array(Question ques[], int capacity) {
     int size = 0;

//The following code does not work, why?
//     string fileNames[] = {"cs135_midterm_f24_v1.txt",
//                           "cs135_midterm_f24_v2.txt",
//                           "cs135_midterm_s24_v1.txt"
//                          };
//
//     int numFiles = sizeof(fileNames, fileNames[0]);
//     for (int i = 0; i < numFiles; i++) {
//         //cout << "file name: " << fileNames[i] << endl;
//         read_file(fileNames[i], ques, capacity, size);
//     }

     read_file("cs135_midterm_f24_v1.txt", ques, capacity, size);
     read_file("cs135_midterm_f24_v2.txt", ques, capacity, size);
     read_file("cs135_midterm_s24_v1.txt", ques, capacity, size);

     string expected = "";
     for (int i = 0; i < size; i++) {
         expected += std::to_string(i + 1) + '\n';
         expected += "question: " + ques[i].text + '\n';
         expected += "answer: " + ques[i].answer + '\n';
         expected += "explanation: " + ques[i].explanation + '\n';
         expected += "type: " + ques[i].type + '\n';
         expected += "version: " + ques[i].version + '\n';
         expected += "label: " + ques[i].label + '\n';
     }

     cout << expected << endl;
}
```

2.12 Define main function

Now the functions are defined and tested, tidy everything up by defining `main` function. The file can be downloaded from **link to incomplete checkAnswer_function_rand.cpp**.

```
#include <iostream>
```

```cpp
#include <fstream> //ifstream
#include <string> //starts_with, c++20
#include <climits> //INT_MAX
#include <string.h> //c-string, strlen(...)
#include <cctype> //isspace

using namespace std;

struct Question {
    string text; //question text
    string answer;
    string explanation;
    string version;
    string type;
    string label;
};

void read_file(string fileName, Question ques[], int capacity, int& size);

void display(Question ques[], int size);

string trim(string str);

//count number of occurrences of ch in str
int count_occurrence(string str, char ch);

//extract type separated by ;
//then put the trimmed type in
//dynamically allocated array of strings
string* extract_type(string type, int& num_types_curr_item);

void insert_order_unique(string types[], int type_capacity, int& type_count, string
   toAdd);

void insert_order_unique(string types[], int type_capacity, int& type_count, Question
   ques[], int ques_size);


void randomize(Question ques[], int size);

string choose_type(string* types, int type_count);

//answer questions, let users try at most 3 times,
//and return the number of correct answers in three or fewer tries
void answer_by_type(Question ques[], int size, string chosenType);

```

```
46 void feedback(int numCorrect, int numQuestionsInType);
47
48 int main() {
49     //Declare CAPACITY as a const int with value 1000.
50     const int CAPACITY = 1000;
51     //Declare ques as an array of Questions
52     //that hold CAPACITY many Questions.
53     Question ques[CAPACITY]; //question array
54
55     //Declare size to be 0.
56     int size = 0;
57     read_file("cs135_midterm_f24_v1.txt", ques, CAPACITY, size);
58     read_file("cs135_midterm_f24_v2.txt", ques, CAPACITY, size);
59     read_file("cs135_midterm_s24_v1.txt", ques, CAPACITY, size);
60
61     //optional
62     //display(ques, size);
63
64     //suppose that there are at most 30 types.
65     const int TYPE_CAPACITY = 30;
66     string types[TYPE_CAPACITY];
67
68     //declare typeCount to be an int with value 0
69     int typeCount = 0;
70
71     //TODO: call insert_order_unique function on questions.
72
73     //TODO: call choose_type function, save the return in a variable.
74
75     //TODO: call answer_by_type with the return from the above statement.
76
77     return 0;
78 }
79
80 //TODO: implement code
81 void read_file(string fileName, Question ques[], int capacity, int& size) {
82     fstream fin(fileName);
83
84     if (fin.fail()) {
85        cerr << fileName << " cannot be opened" << endl;
86        exit(1);
87     }
88
89     string text;
90     string answer;
91     string explanation;
```

```
92      string version;
93      string type; //type of the code
94      string label;
95
96      string line;
97
98  //skip lines until get the first question
99      while ( getline(fin, line) && !( line.starts_with("question: ") || line.starts_with
        ("Question: ") ) )
100         ;
101
102     while (line.starts_with("question: ")) {
103         text = line.substr(strlen("question: "));
104
105         line = "";
106         while (getline(fin, line) && !line.starts_with("answer: ") )
107             text += '\n' + line;
108
109         if ( line.starts_with("answer: ") ) {
110            answer = line.substr(strlen("answer: "));
111
112            line = "";
113            while (getline(fin, line) && !(line.starts_with("question: ") || line.
       starts_with("version: ") || line.starts_with("label: ") || line.starts_with("type: "
       ) || line.starts_with("explanation: ") ) )
114                answer += line + '\n';
115
116            //explanation is the next entry following answer.
117            //Need to starts with "explanation: ",
118            //cannot handle the case like "explanation:\n"
119            if ( line.starts_with("explanation: ") ) {
120               explanation = line.substr(strlen("explanation: ")) + '\n';
121               line = "";
122               while (getline(fin, line) && !(line.starts_with("question: ") || line.
       starts_with("version: ") || line.starts_with("label: ") || line.starts_with("type: "
       ) ) )
123                   explanation += line + '\n';
124            }
125
126            //use do-while statement, otherwise, the entry following answer: is not read
127            do {
128                if (line.starts_with("version: "))
129                   version = line.substr(strlen("version: "));
130                else if (line.starts_with("type: "))
131                       type = line.substr(strlen("type: "));
132                else if (line.starts_with("label: "))
```

```
                        label = line.substr(strlen("label: "));
                line = "";
            } while (getline(fin, line) && !(line.starts_with("question: ")));

            //TODO: if size is larger or equal to capacity, close the file and return.


            //TODO: save text, answer, ..., explantion, to the corresponding field of
    ques[size].


            //TODO: increase size by 1


            text = "";
            answer = "";
            version = "";
            type = "";
            label = "";
            explanation = "";
        }
    }
    fin.close();
}

void display(Question ques[], int size) {
    for (int i = 0; i < size; i++) {
        cout << i + 1 << endl;
        cout << "question: " << ques[i].text << endl;
        cout << "answer: " << ques[i].answer << endl;
        cout << "explanation: " << ques[i].explanation << endl;
        cout << "type: " << ques[i].type <<  endl;
        cout << "version: " << ques[i].version <<  endl;
        cout << "label: " << ques[i].label <<  endl;
        cout << endl;
    }
}

//TODO: implement code
string trim(string str) {

}

//TODO: implement code
//count number of occurrences of ch in str
int count_occurrences(string str, char ch) {
```

```
178
179 }
180
181 //TODO: implement code
182 string* extract_type(string type, int& num_types_curr_item) {
183
184 }
185
186 //TODO: implement code
187 void insert_order_unique(string types[], int type_capacity, int& size, string toAdd) {
188
189 }
190
191 //TODO: implement code
192 void insert_order_unique(string types[], int type_capacity, int& numTypes, Question ques
        [], int ques_size) {
193
194 }
195
196 //TODO: implement code
197 string choose_type(string* types, int type_count) {
198
199 }
200
201 //TODO: implement code
202 void answer_by_type(Question ques[], int size, string chosenType) {
203     //TODO: call randomize function
204
205     //TODO: display questions and answer them
206     //If fail to answer a question correctly in 3 tries,
207     //if explanation field is not empty,
208     //display explantion field of that question.
209
210     //TODO: call feedback function
211
212 }
213
214 //TODO: implement code
215 void feedback(int numCorrect, int numQuestions) {
216
217 }
218
219 //TODO: implement code
220 void randomize(Question ques[], int size) {
221
222 }
```