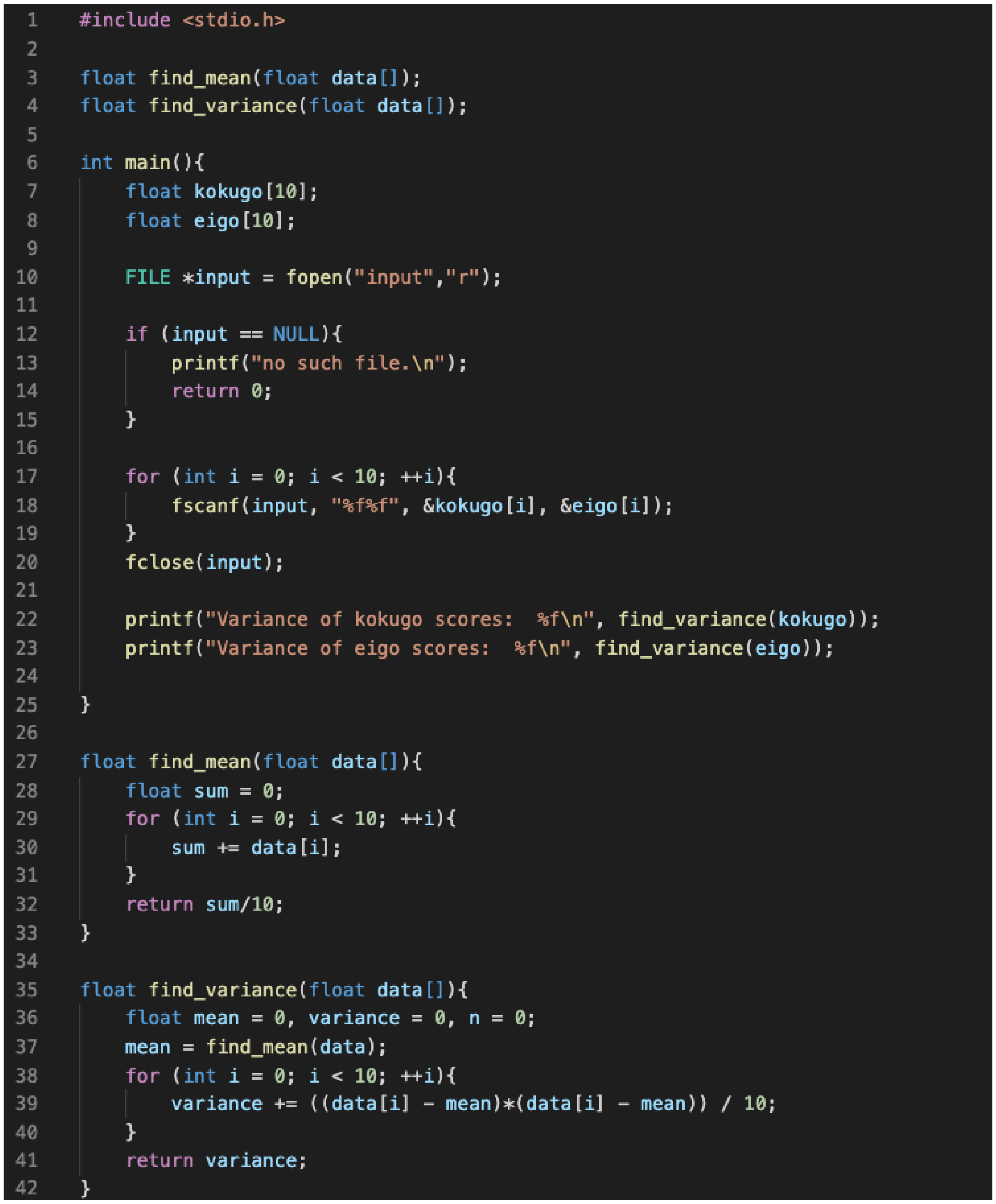
202018021
Fu, Ziyu
Programming I Chapter 7 Task 1, 2, 3
2021/06/10

Task 1

1. Objective
   To make a function that calculates the variance of a set of data in an array of length 10.

2. Strategy of solving
   - Read and understand the task statement
   - Design the code
   - Observe the output

3. Program code



```c
#include <stdio.h>

float find_mean(float data[]);
float find_variance(float data[]);

int main(){
    float kokugo[10];
    float eigo[10];

    FILE *input = fopen("input","r");

    if (input == NULL){
        printf("no such file.\n");
        return 0;
    }

    for (int i = 0; i < 10; ++i){
        fscanf(input, "%f%f", &kokugo[i], &eigo[i]);
    }
    fclose(input);

    printf("Variance of kokugo scores:  %f\n", find_variance(kokugo));
    printf("Variance of eigo scores:  %f\n", find_variance(eigo));

}

float find_mean(float data[]){
    float sum = 0;
    for (int i = 0; i < 10; ++i){
        sum += data[i];
    }
    return sum/10;
}

float find_variance(float data[]){
    float mean = 0, variance = 0, n = 0;
    mean = find_mean(data);
    for (int i = 0; i < 10; ++i){
        variance += ((data[i] - mean)*(data[i] - mean)) / 10;
    }
    return variance;
}
```

4. Results and discussions

   <u>Console output:</u>

```
Variance of kokugo scores:  561.840027
Variance of eigo scores:  427.210022
```

The program outputs the correct results. The output numbers are subject to floating point rounding error but that is expected.

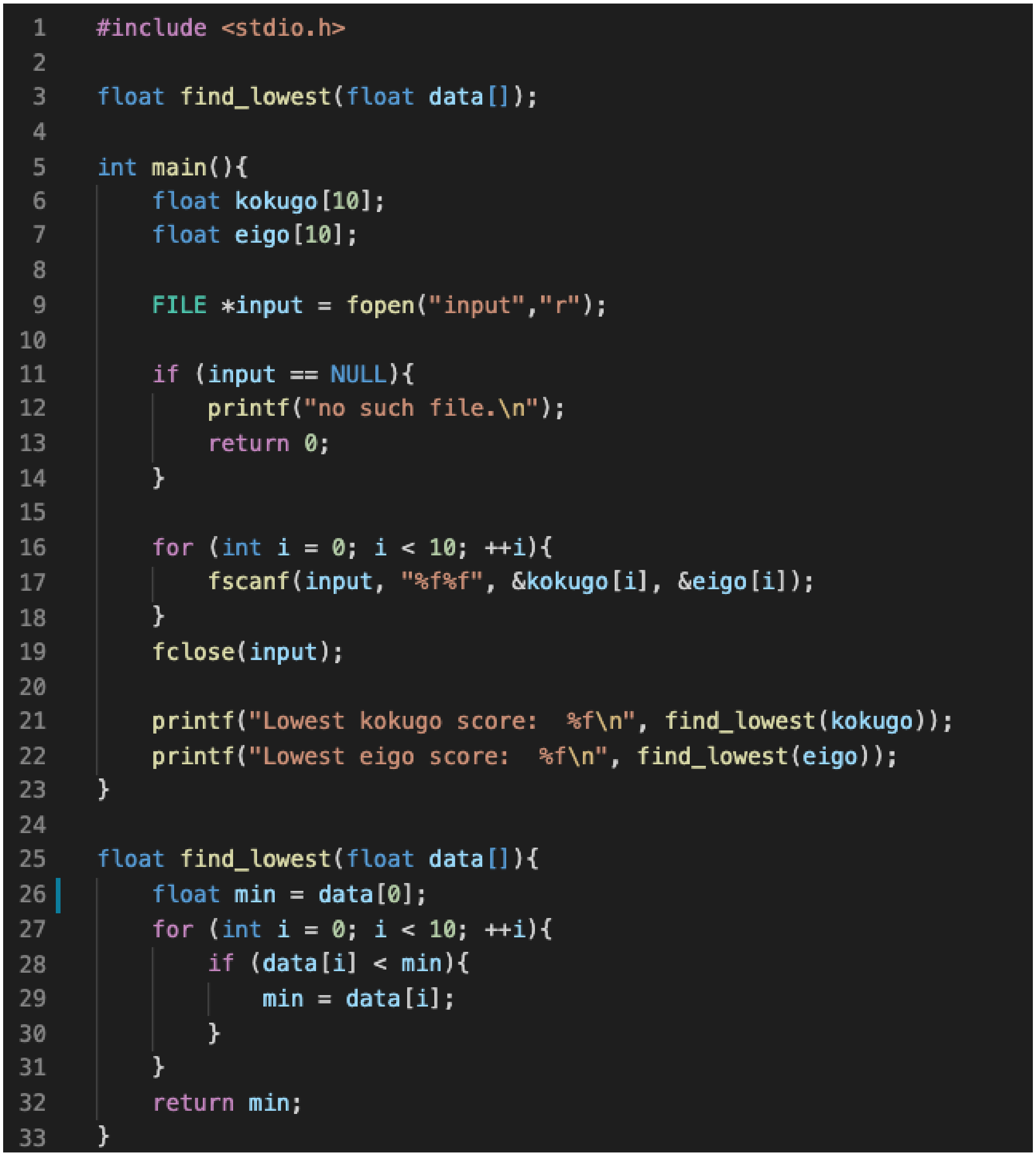
I used **`fscanf`** to read a **`FILE`** object instead of taking the data using **`scanf`** from user input. It seemed more elegant to me. However, the assumption that all arrays have length = 10 runs throughout the program. If the input file didn't only have 10 people's worth of data in it, the array would need to be a different length, in which case it might have been better to explicitly pass the length of the arrays into the functions as well. (Maybe use a counter to automatically count how many lines in the file have valid data??)

<u>Task 2</u>

1. Objective
   To create a function that finds the minimum value in an array on length 10.

2. Strategy of solving
   - Read and understand the task statement
   - Design the code
   - Observe the outputs

3. Program code



```c
#include <stdio.h>

float find_lowest(float data[]);

int main(){
    float kokugo[10];
    float eigo[10];

    FILE *input = fopen("input","r");

    if (input == NULL){
        printf("no such file.\n");
        return 0;
    }

    for (int i = 0; i < 10; ++i){
        fscanf(input, "%f%f", &kokugo[i], &eigo[i]);
    }
    fclose(input);

    printf("Lowest kokugo score:  %f\n", find_lowest(kokugo));
    printf("Lowest eigo score:  %f\n", find_lowest(eigo));
}

float find_lowest(float data[]){
    float min = data[0];
    for (int i = 0; i < 10; ++i){
        if (data[i] < min){
            min = data[i];
        }
    }
    return min;
}
```

4. Results and discussions

   Console output:

```
Lowest kokugo score:  24.000000
Lowest eigo score:  34.000000
```

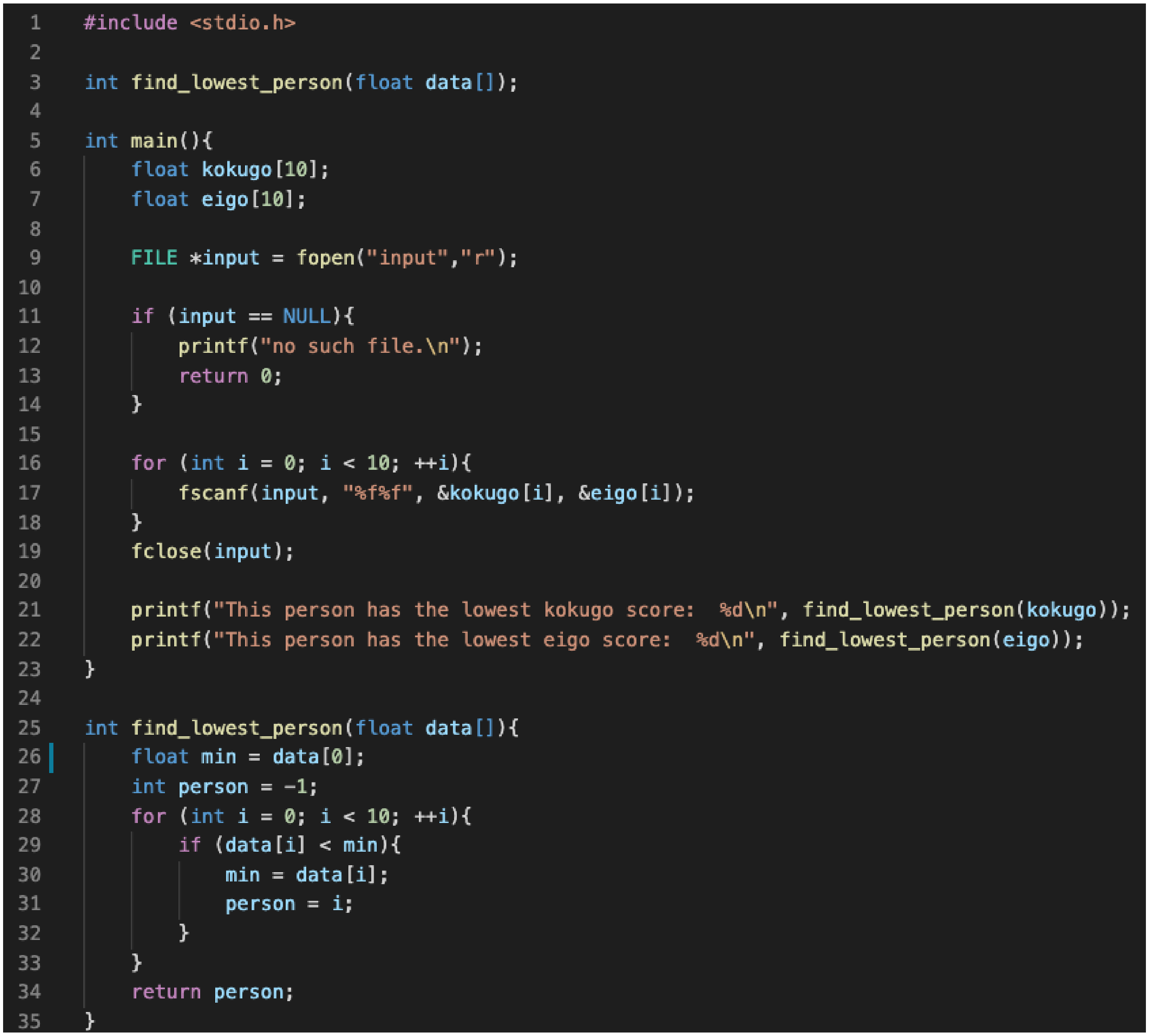
   The program outputs the correct results as expected. The assumption on array length holds in this program too. It will break if the input file has more than 10 lines of data.

Task 3

1. Objective
   To create a function that finds at which position does the minimum value occur in an array of length 10.

2. Strategy of solving
   - Read and understand the task statement
   - Design the code
   - Observe the outputs

3. Program code



```c
#include <stdio.h>

int find_lowest_person(float data[]);

int main(){
    float kokugo[10];
    float eigo[10];

    FILE *input = fopen("input","r");

    if (input == NULL){
        printf("no such file.\n");
        return 0;
    }

    for (int i = 0; i < 10; ++i){
        fscanf(input, "%f%f", &kokugo[i], &eigo[i]);
    }
    fclose(input);

    printf("This person has the lowest kokugo score:  %d\n", find_lowest_person(kokugo));
    printf("This person has the lowest eigo score:  %d\n", find_lowest_person(eigo));
}

int find_lowest_person(float data[]){
    float min = data[0];
    int person = -1;
    for (int i = 0; i < 10; ++i){
        if (data[i] < min){
            min = data[i];
            person = i;
        }
    }
    return person;
}
```

4. Results and discussions

Console output:

```
This person has the lowest kokugo score:  3
This person has the lowest eigo score:  2
```

The program outputs the correct results as expected. This is a very simple variation on the task 2 code. The assumption on array length obviously still holds in this program as well. Aside from that, this program also assumes that the lowest score can only occur once in each case. It will only report the first occurrence of the lowest number in each search as the condition is set to “smaller than”.