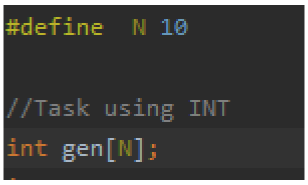
# Task 1: Problem solving

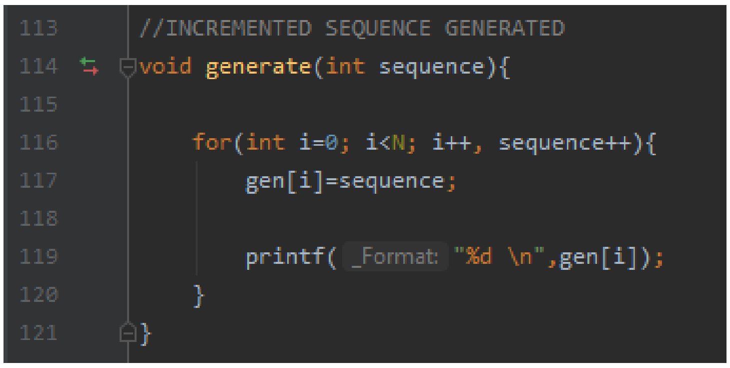
## Task (a)(i):

*Generate(): a function that populates an array with a sequence of N integers, in ascending order, and starting off integer i having a minimum value of 1. i should be accepted as a function argument while N is defined as a constant. This function is destructive in the sense that it overwrites any previously generated values.*

To solve the problem, first N has to be defined, in this case giving it the value of 10 which is subject to change as the programmer requires. An array of size N has also been created but left empty, in order to be filled with the sequence that will be generated. These declarations are all found in the header file.



```c
#define  N 10

//Task using INT
int gen[N];
```



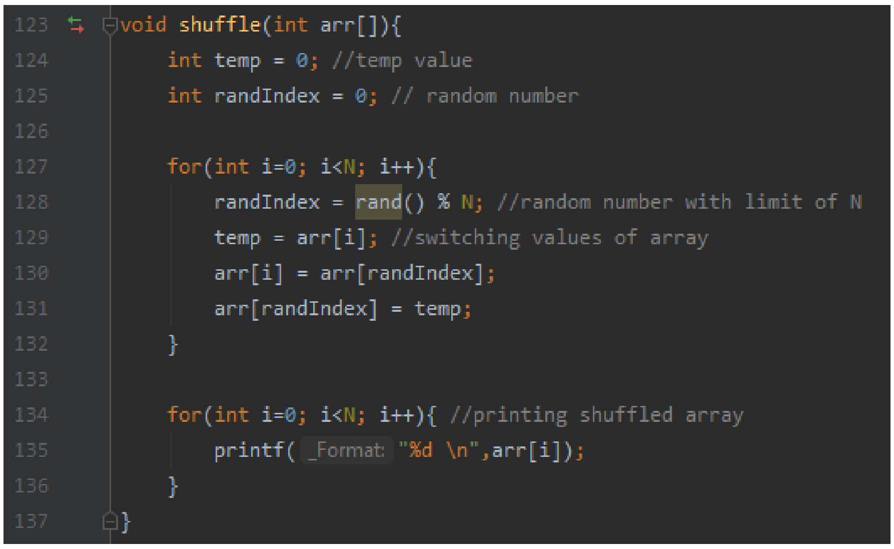
```c
//INCREMENTED SEQUENCE GENERATED
void generate(int sequence){

    for(int i=0; i<N; i++, sequence++){
        gen[i]=sequence;

        printf( _Format: "%d \n",gen[i]);
    }
}
```

A function generate is created. Its purpose is to take an integer given by the user which will be used as the sequence's starting point. The first number of the sequence (named sequence in the function) is passed as an argument. It then increments the value for N times, thus generating a sequence in ascending order. It can also print out said sequence.

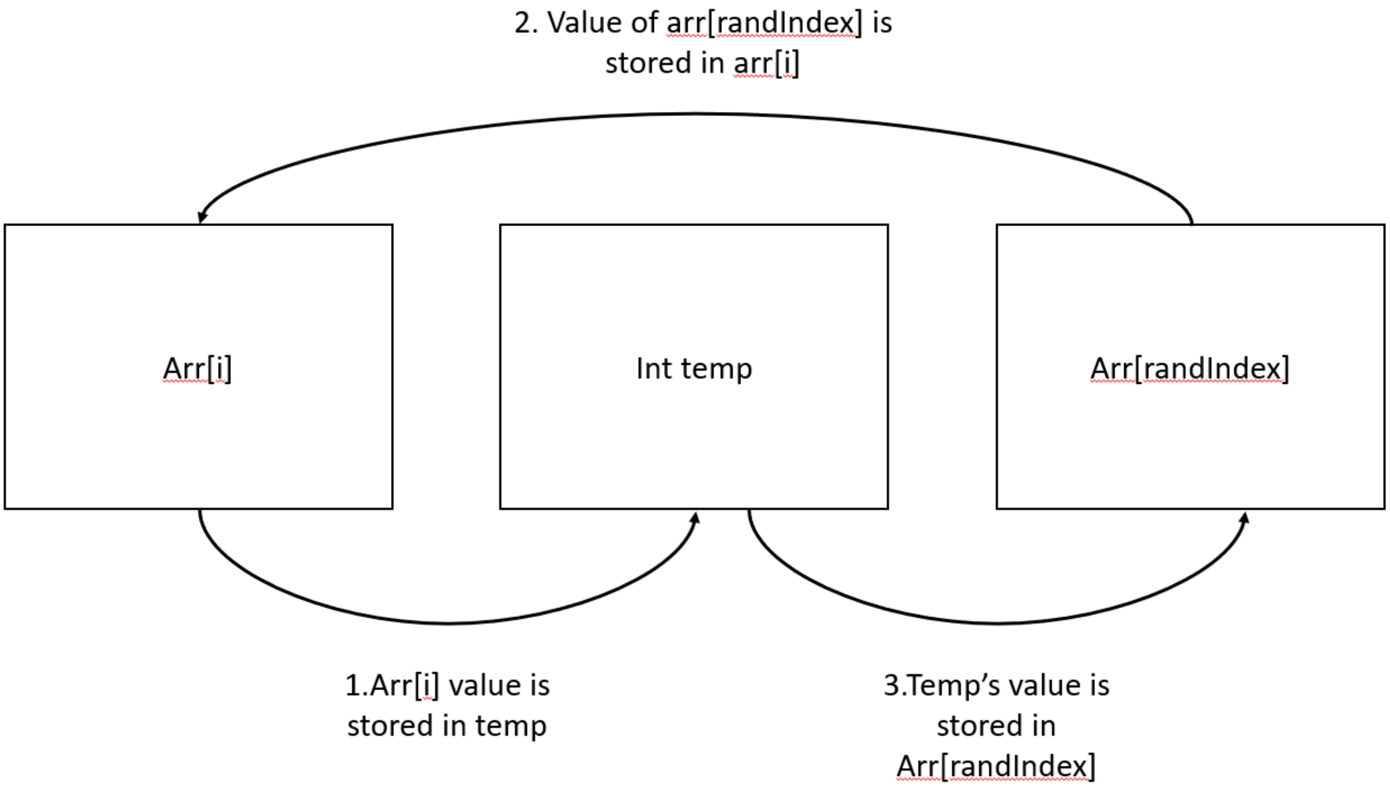
## Task (a)(ii):

*shuffle(): a function that shuffles the items of an array argument and which makes use of stdlib.h's rand() function.*

The function starts by declaring int temp and int randIndex while giving them the value of 0. An array is passed through an argument which will be adjusted by the function. A for loop is started which occurs for N times. In this loop, randIndex is given a random number using stdlib.h's rand() function and restricted by the value of N. Furthermore, the values of the number in the array are shuffled around as shown in the step by step diagram below. When the function is done and all the values are shuffled, another for loop starts which prints the shuffled array.



```c
void shuffle(int arr[]){
    int temp = 0; //temp value
    int randIndex = 0; // random number

    for(int i=0; i<N; i++){
        randIndex = rand() % N; //random number with limit of N
        temp = arr[i]; //switching values of array
        arr[i] = arr[randIndex];
        arr[randIndex] = temp;
    }

    for(int i=0; i<N; i++){ //printing shuffled array
        printf( _Format: "%d \n",arr[i]);
    }
}
```


2. Value of arr[randIndex] is stored in arr[i]
Arr[i]
Int temp
Arr[randIndex]
1.Arr[i] value is stored in temp
3.Temp's value is stored in Arr[randIndex]

## Task (a)(iii):

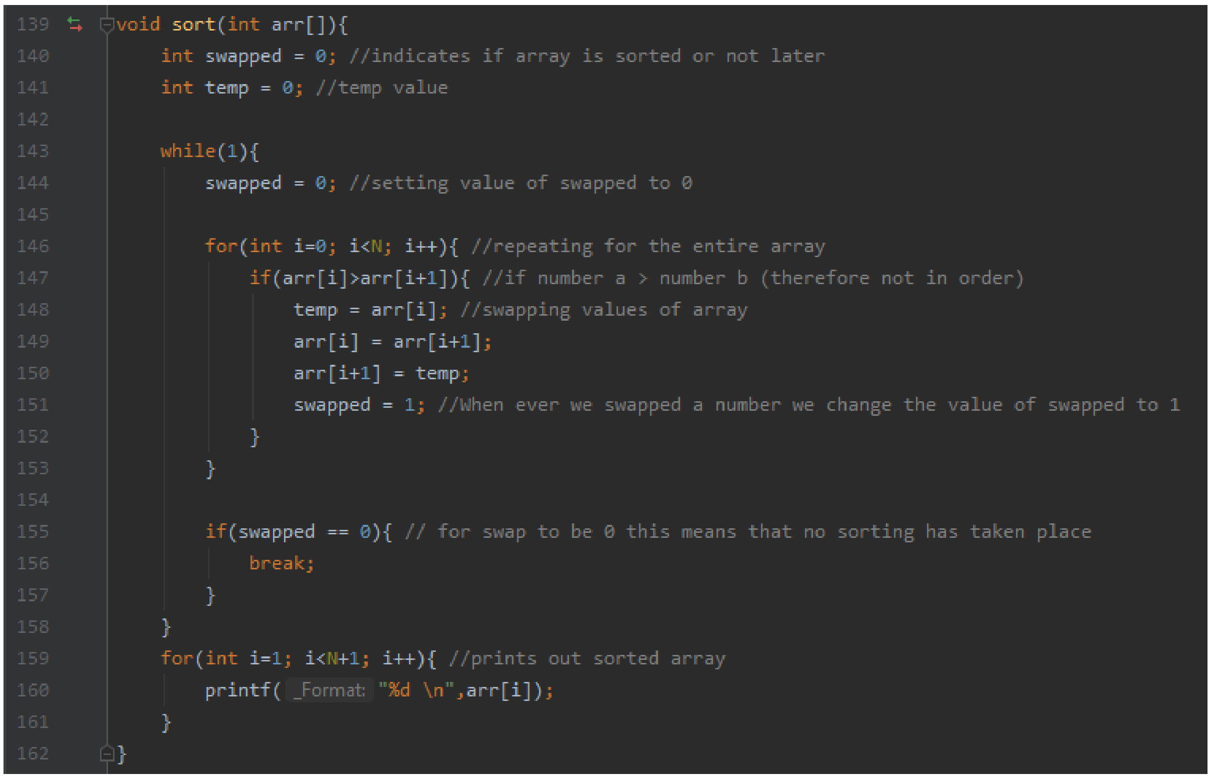
*sort(): a function (implemented from scratch) that returns a sorted array passed as an argument.*

Bubble sort was used to sort the sequence. The function starts by declaring “int swapped” and giving it the value of 0. This integer is vital in this function as with it we can determine whether the array is sorted or not (further detail given below). An array is passed through an argument which will be adjusted by the function. In order to swap the array values around we use the same concept as explained in the shuffle function (using temp). The difference is that the swapping only occurs if a number is bigger than the one next to it. The concept is show below:

7, 5, 10, ……

5, 7, 10, ……

When a function is swapped, the value of “int swapped” is changed to 1. The function will keep on sorting until the value of “int swapped” remains as 0, meaning that no swap has occurred and thus the array is finally sorted. At the end the function also prints out the sorted array.

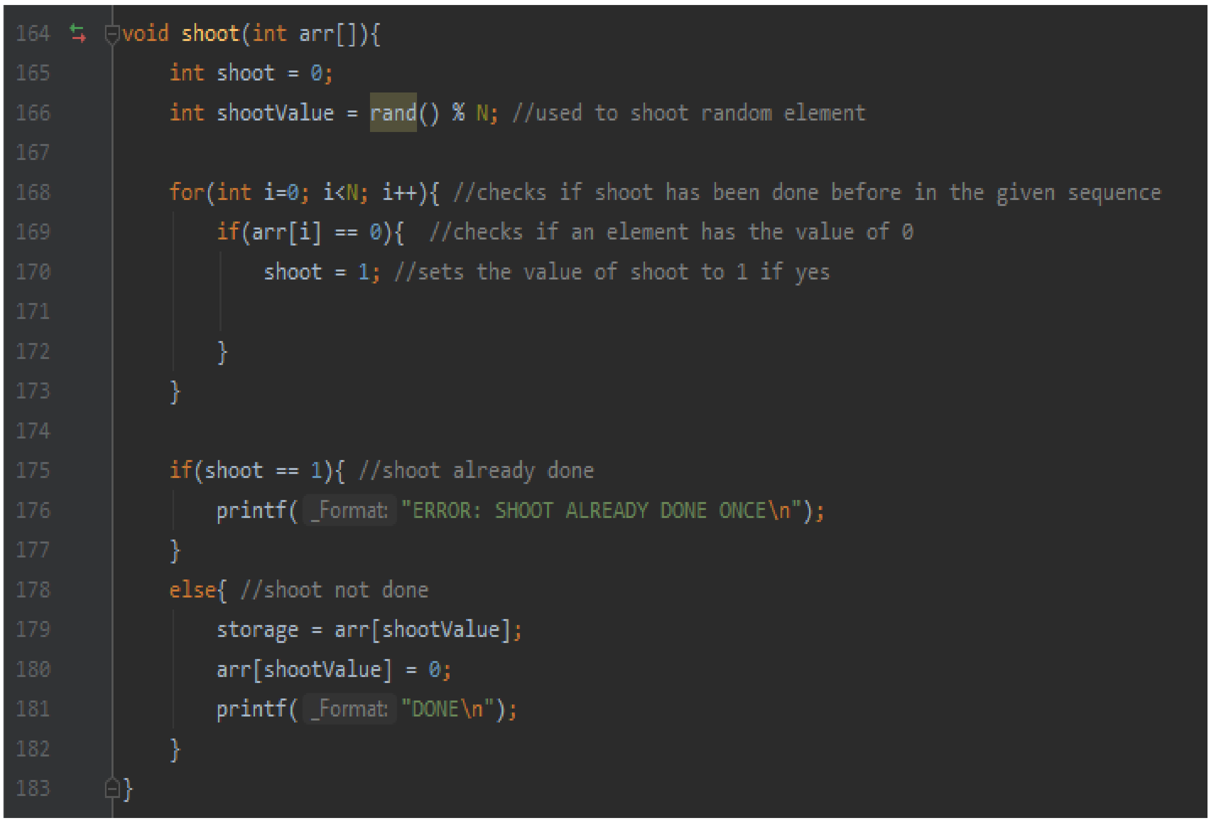


```c
void sort(int arr[]){
    int swapped = 0; //indicates if array is sorted or not later
    int temp = 0; //temp value

    while(1){
        swapped = 0; //setting value of swapped to 0

        for(int i=0; i<N; i++){ //repeating for the entire array
            if(arr[i]>arr[i+1]){ //if number a > number b (therefore not in order)
                temp = arr[i]; //swapping values of array
                arr[i] = arr[i+1];
                arr[i+1] = temp;
                swapped = 1; //When ever we swapped a number we change the value of swapped to 1
            }
        }

        if(swapped == 0){ // for swap to be 0 this means that no sorting has taken place
            break;
        }
    }
    for(int i=1; i<N+1; i++){ //prints out sorted array
        printf( _Format: "%d \n",arr[i]);
    }
}
```

## Task (a)(iv):

*shoot(): a function that zeros out one element from, a possibly unsorted, array at random. This function returns an error if at least one element had already been previously zeroed out.*

The function shoot starts by declaring “int shoot” and giving it the value of 0. An array is passed through an argument which will be adjusted by the function. It also declares “int shootValue” and gives it a random number (using the rand function). The function will ‘shoot’ an element from the array and give it the value of 0. However, it also stores the original value in int storage, declared outside of the function for use elsewhere. When done it prints out “DONE” in order to indicate that the swap has worked as intended. The error still must be coded in which is where “int shoot” comes in. Before the swapping occurs a for loop scans the array to check if there is a 0 present. Since the minimum value of the sequence in 1, the only way for there to be a 0 is through the shoot function. If no 0 is found, then nothing happens, and the code goes on as normal. If a 0 is found, shoot’s value is changed to 1, and thanks to the if else statement after it, the swapping will not take place if shoot’s value is equal to 1 and it will instead output an ERROR message. Thus, if a number in the generated sequence has been “shot”, then an error will occur, and it won’t be shot again.

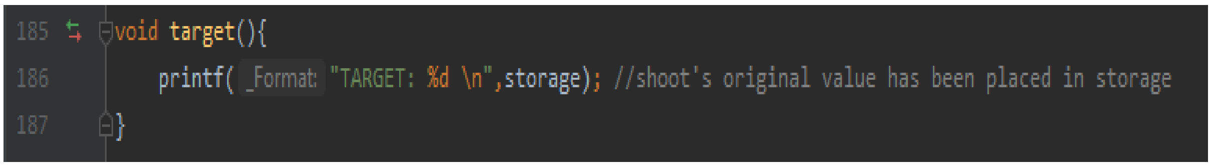


```
void shoot(int arr[]){
    int shoot = 0;
    int shootValue = rand() % N; //used to shoot random element

    for(int i=0; i<N; i++){ //checks if shoot has been done before in the given sequence
        if(arr[i] == 0){  //checks if an element has the value of 0
            shoot = 1; //sets the value of shoot to 1 if yes

        }
    }

    if(shoot == 1){ //shoot already done
        printf( _Format: "ERROR: SHOOT ALREADY DONE ONCE\n");
    }
    else{ //shoot not done
        storage = arr[shootValue];
        arr[shootValue] = 0;
        printf( _Format: "DONE\n");
    }
}
```

## Task (a)(v):

*target(): a function that returns the number (i.e. the actual value and not the array offset) that was zeroed out by a single call to shoot().*

The function target simply outputs “int storage” which contains element that was “shot” from the shoot function above.

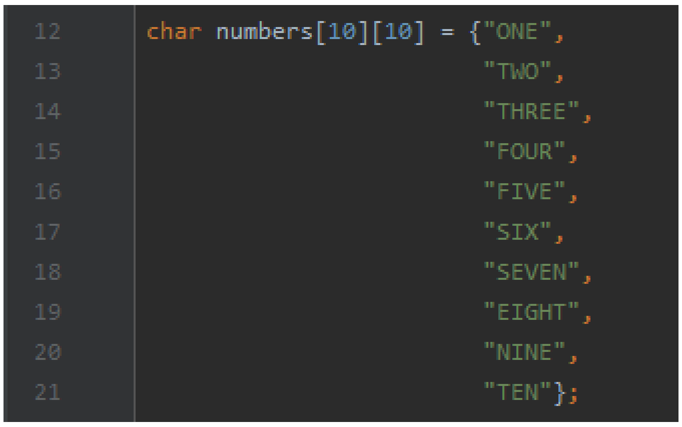


```
void target(){
    printf( _Format: "TARGET: %d \n",storage); //shoot's original value has been placed in storage
}
```

## Task b

*Implement a string array version for each of the above functions. In this case restrict the range of array values to just the string representations of the integers from "one" to "ten". Make sure to provide an optimized implementation despite the concise strings involved.*

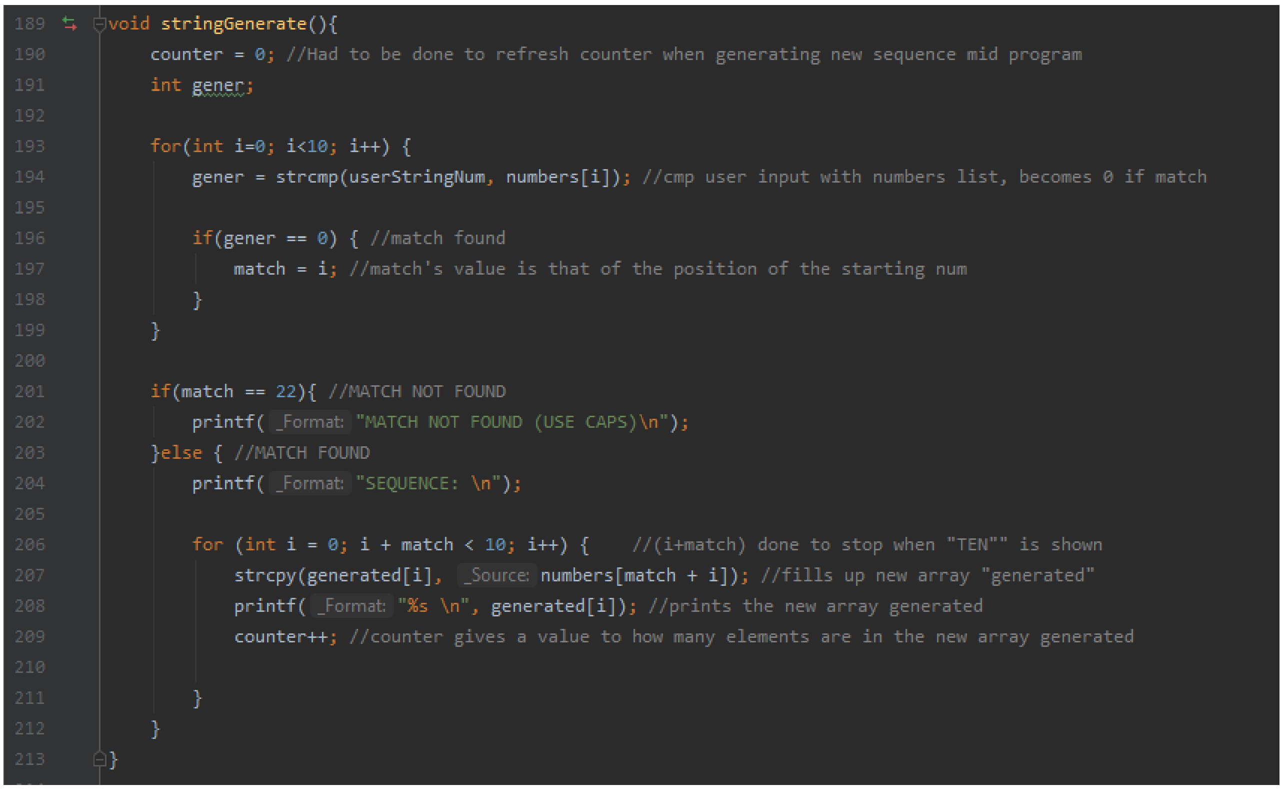
To complete Task 2, one had to create functions like Task 1, but ones which are compatible with characters.
To start with, a 3D array was created, containing the range of numbers given in the task:



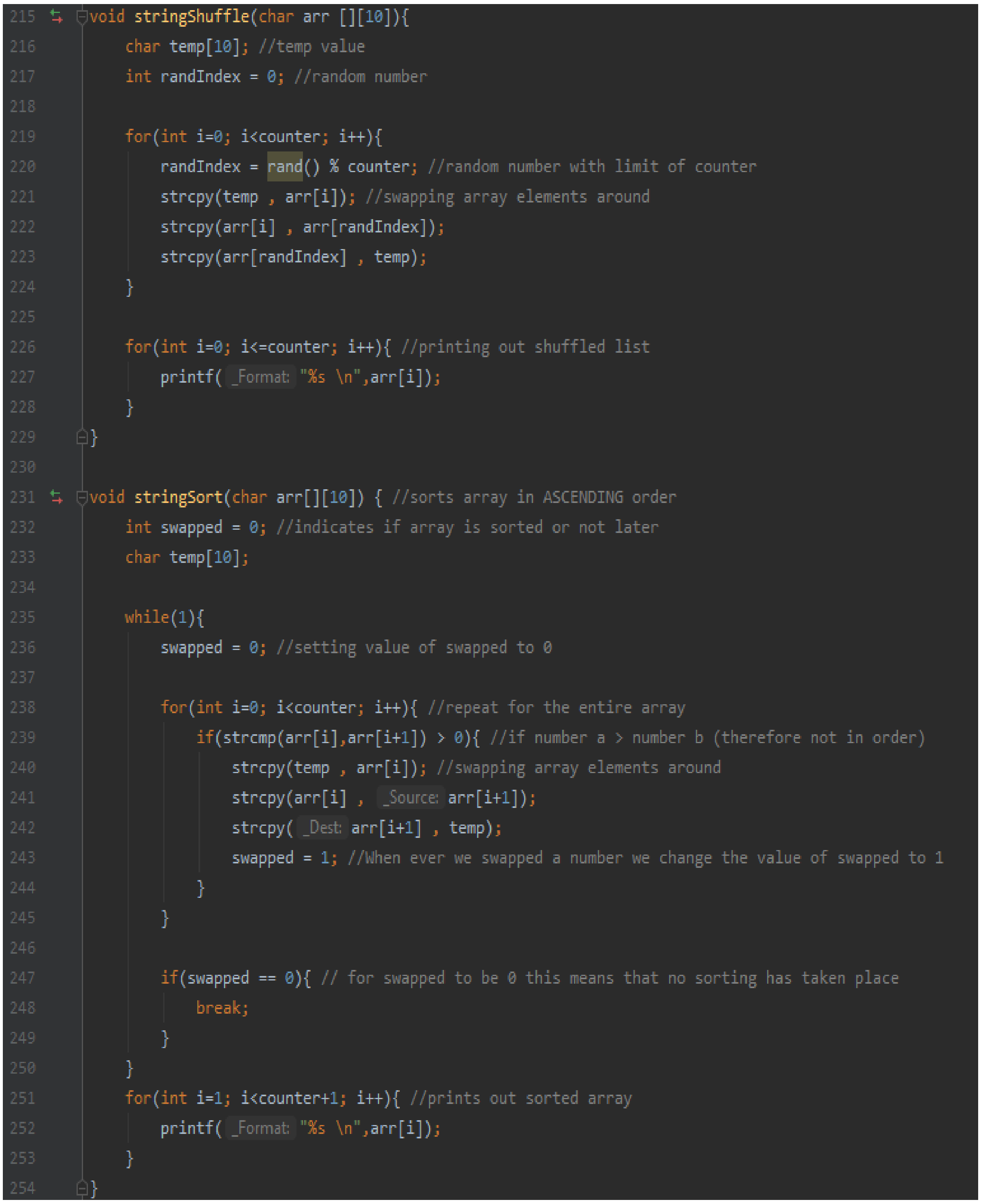
```
char numbers[10][10] = {"ONE",
                        "TWO",
                        "THREE",
                        "FOUR",
                        "FIVE",
                        "SIX",
                        "SEVEN",
                        "EIGHT",
                        "NINE",
                        "TEN"};
```

Secondly function stringGenerate() was created. This function's purpose is to take a user's input, match it to the array of character numbers created earlier, and output the rest of the sequence starting from the user's input. To do this, first "int gener" was declared. Next a for loop was created which cycles through the array numbers. If the user's input is matched with the array (through the strcmp), the value of "gener" becomes 0. If this does occur, then integer match (declared outside of the function) has its value equal to the position of said match. Finally, another for loop is installed, which puts the values of the sequence in a new array called "generated". It also prints the array. This was done to create a new array which starts from the user's input, making the next tasks easier. A counter is also present to know the new array's length. This new counter integer will be used instead of the integer N. This was done by incrementing the integer counter by 1 for every value placed in the array "generated". Furthermore, at the beginning of the function, counter's value is reset to 0, in order to repeat the function numerous times without any conflicts.

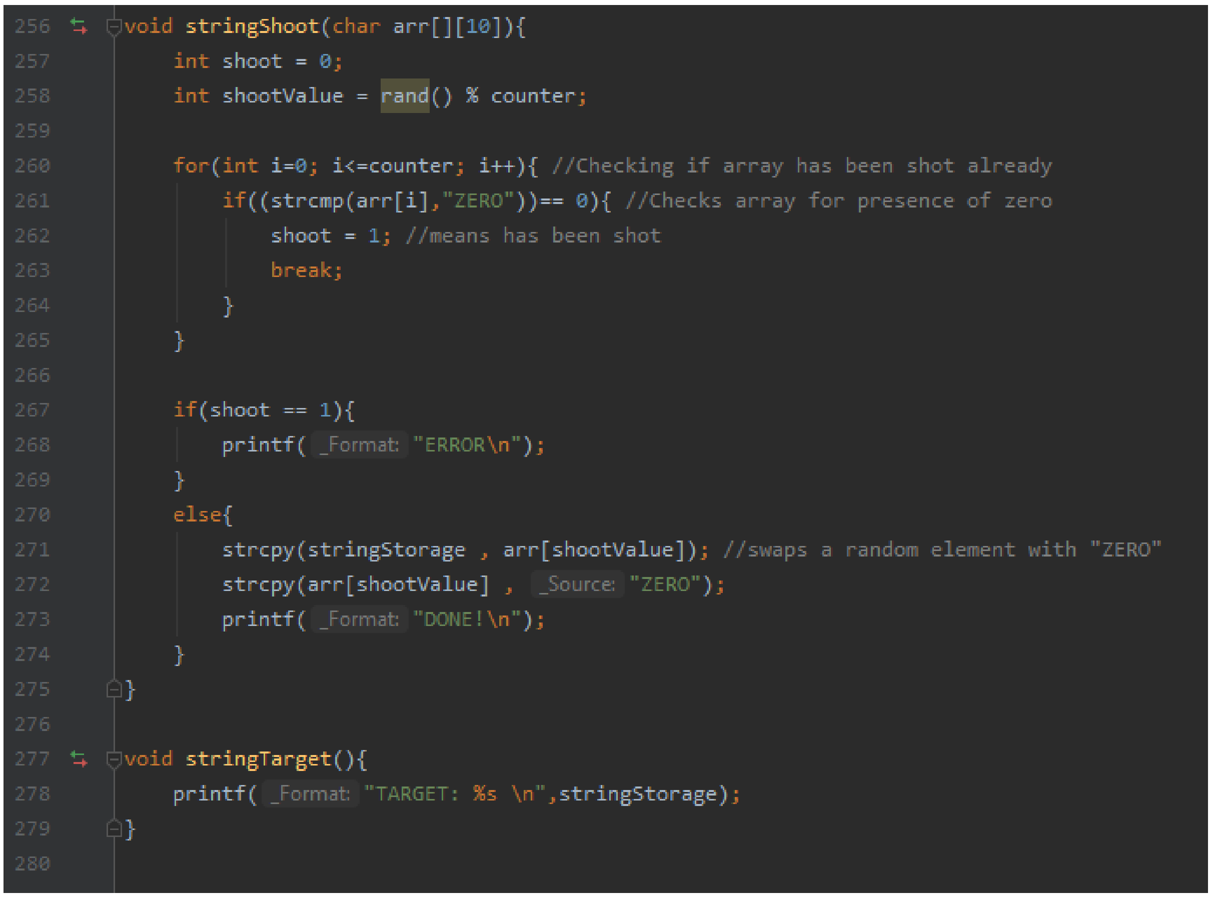
However, should use user's input not be matched with the created 3D array, the value of "match" would stay equal to 22 (assigned a random number as long as it's bigger then 10), and a "MATCH NOT FOUND" error would instead be outputted.


```
void stringGenerate(){
    counter = 0; //Had to be done to refresh counter when generating new sequence mid program
    int gener;

    for(int i=0; i<10; i++) {
        gener = strcmp(userStringNum, numbers[i]); //cmp user input with numbers list, becomes 0 if match

        if(gener == 0) { //match found
            match = i; //match's value is that of the position of the starting num
        }
    }

    if(match == 22){ //MATCH NOT FOUND
        printf( _Format: "MATCH NOT FOUND (USE CAPS)\n");
    }else { //MATCH FOUND
        printf( _Format: "SEQUENCE: \n");

        for (int i = 0; i + match < 10; i++) {    //(i+match) done to stop when "TEN"" is shown
            strcpy(generated[i],  _Source: numbers[match + i]); //fills up new array "generated"
            printf( _Format: "%s \n", generated[i]); //prints the new array generated
            counter++; //counter gives a value to how many elements are in the new array generated

        }
    }
}
```

The following functions “stringSuffle”, “stringSort”, “stringShoot” and “stringTarget” use the same methods and concepts as their integer counterparts but have techniques such as strcpy and strcmp which must be used in cases like these. The counter plays the role of N (the array’s max length). This value of counter is always changing and could not be predeclared because its value is constantly shifting depending on the sequences starting point.



```c
void stringShuffle(char arr [][10]){
    char temp[10]; //temp value
    int randIndex = 0; //random number

    for(int i=0; i<counter; i++){
        randIndex = rand() % counter; //random number with limit of counter
        strcpy(temp , arr[i]); //swapping array elements around
        strcpy(arr[i] , arr[randIndex]);
        strcpy(arr[randIndex] , temp);
    }

    for(int i=0; i<=counter; i++){ //printing out shuffled list
        printf( _Format: "%s \n",arr[i]);
    }
}

void stringSort(char arr[][10]) { //sorts array in ASCENDING order
    int swapped = 0; //indicates if array is sorted or not later
    char temp[10];

    while(1){
        swapped = 0; //setting value of swapped to 0

        for(int i=0; i<counter; i++){ //repeat for the entire array
            if(strcmp(arr[i],arr[i+1]) > 0){ //if number a > number b (therefore not in order)
                strcpy(temp , arr[i]); //swapping array elements around
                strcpy(arr[i] , _Source: arr[i+1]);
                strcpy( _Dest: arr[i+1] , temp);
                swapped = 1; //When ever we swapped a number we change the value of swapped to 1
            }
        }

        if(swapped == 0){ // for swapped to be 0 this means that no sorting has taken place
            break;
        }
    }
    for(int i=1; i<counter+1; i++){ //prints out sorted array
        printf( _Format: "%s \n",arr[i]);
    }
}
```

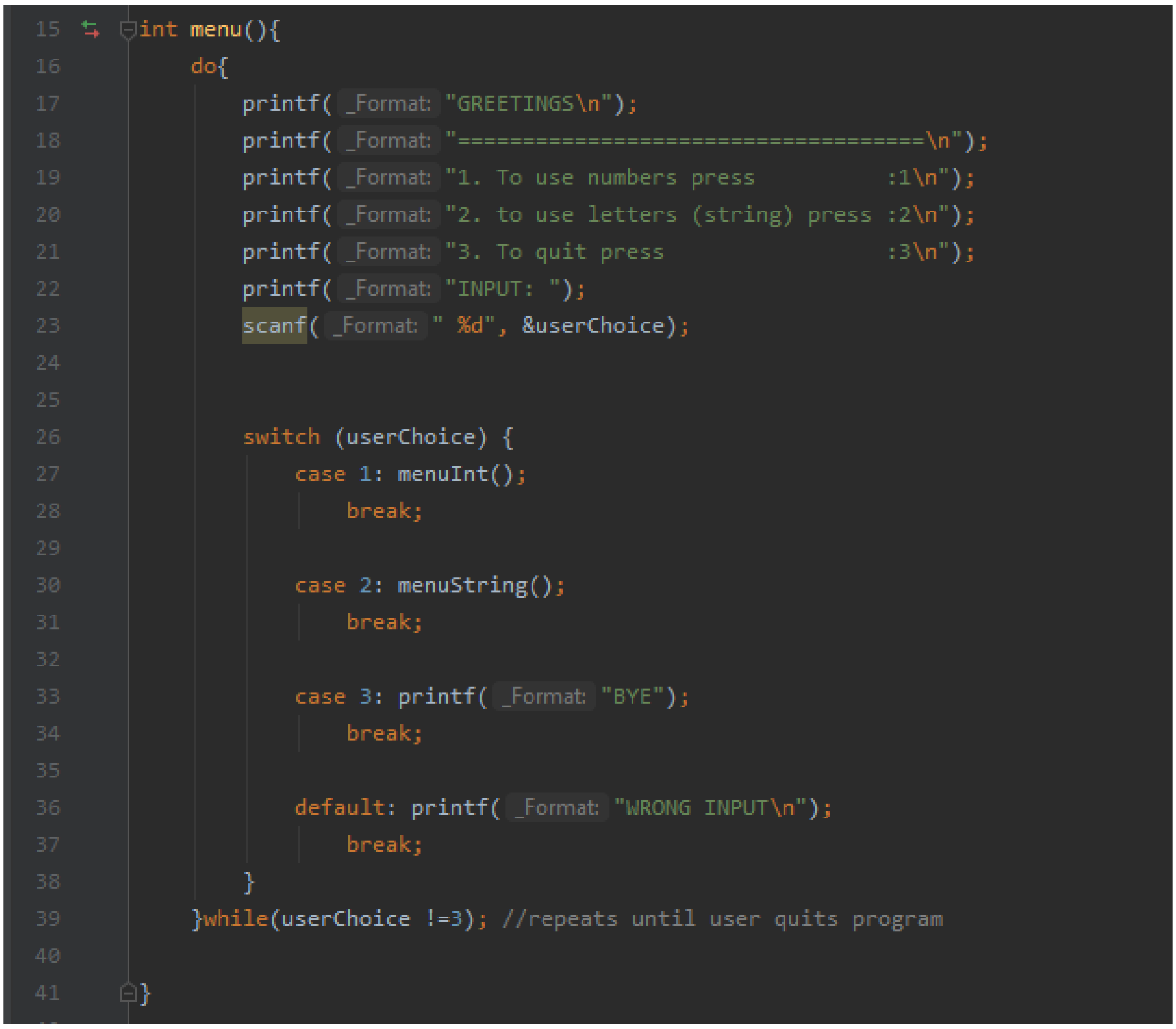


```c
void stringShoot(char arr[][10]){
    int shoot = 0;
    int shootValue = rand() % counter;

    for(int i=0; i<=counter; i++){ //Checking if array has been shot already
        if((strcmp(arr[i],"ZERO"))== 0){ //Checks array for presence of zero
            shoot = 1; //means has been shot
            break;
        }
    }

    if(shoot == 1){
        printf( _Format: "ERROR\n");
    }
    else{
        strcpy(stringStorage , arr[shootValue]); //swaps a random element with "ZERO"
        strcpy(arr[shootValue] ,  _Source: "ZERO");
        printf( _Format: "DONE!\n");
    }
}

void stringTarget(){
    printf( _Format: "TARGET: %s \n",stringStorage);
}
```

## Task c

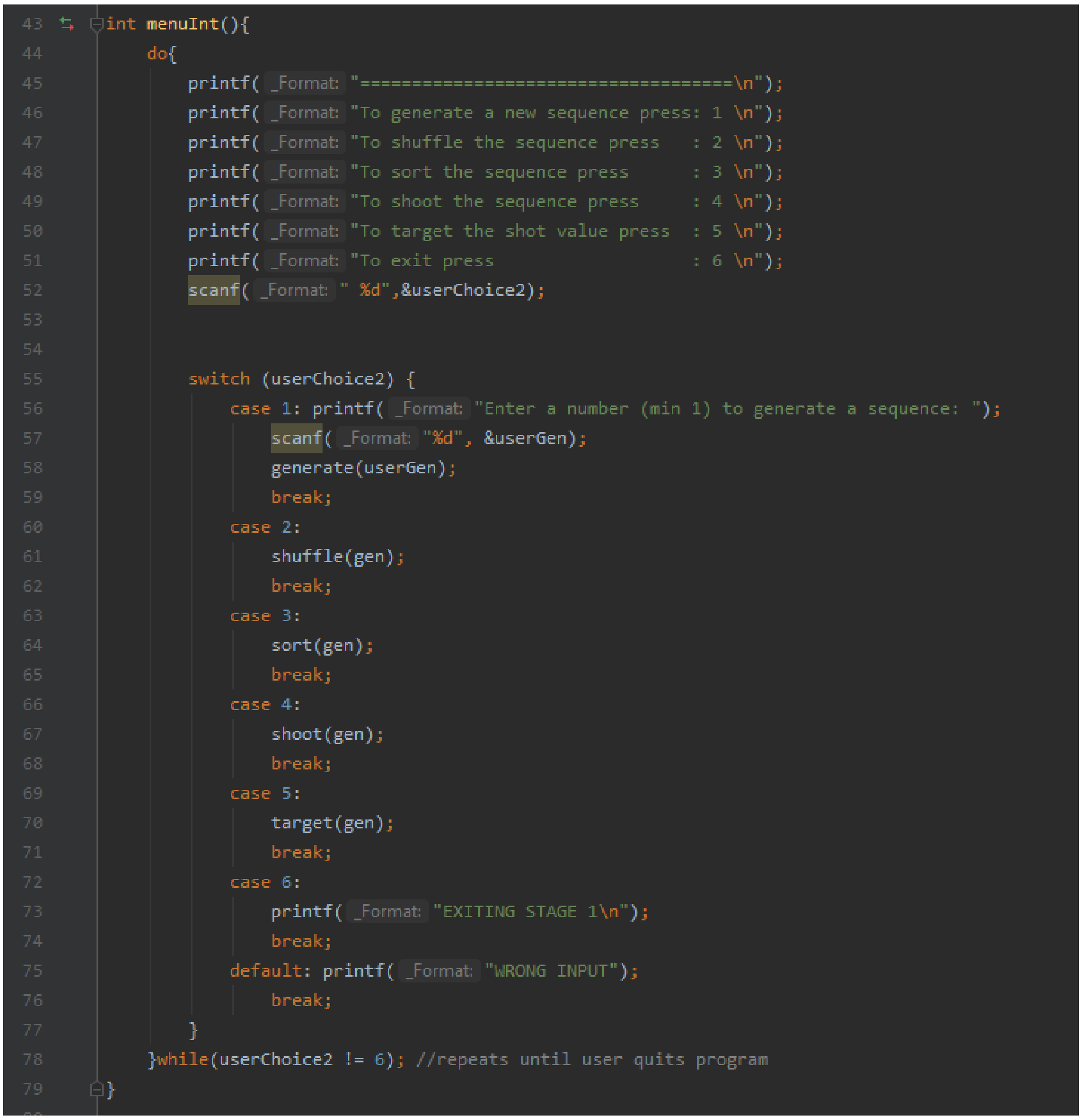
*Write a program that presents the end-user with a command-line menu, and that repeatedly asks the user to execute either of the functions above, or to quit. The program should prompt for function arguments accordingly and the sole manner by which arrays should be populated is by calling generate(). Proper validation of user input is expected.*

The first step of the menu was to prompt the user with the option of using either numbers, letters (string) or to exit the program. This was done using switch cases. A do while loop is implemented with the switch in order to keep the program running until the user enters in the option to exit the program. In case 1 (to use integers) a function menuInt() is called. A nested switch case could have been implemented instead but I found this method to be less messy. The same thing is done with the string version but menuString() is called.

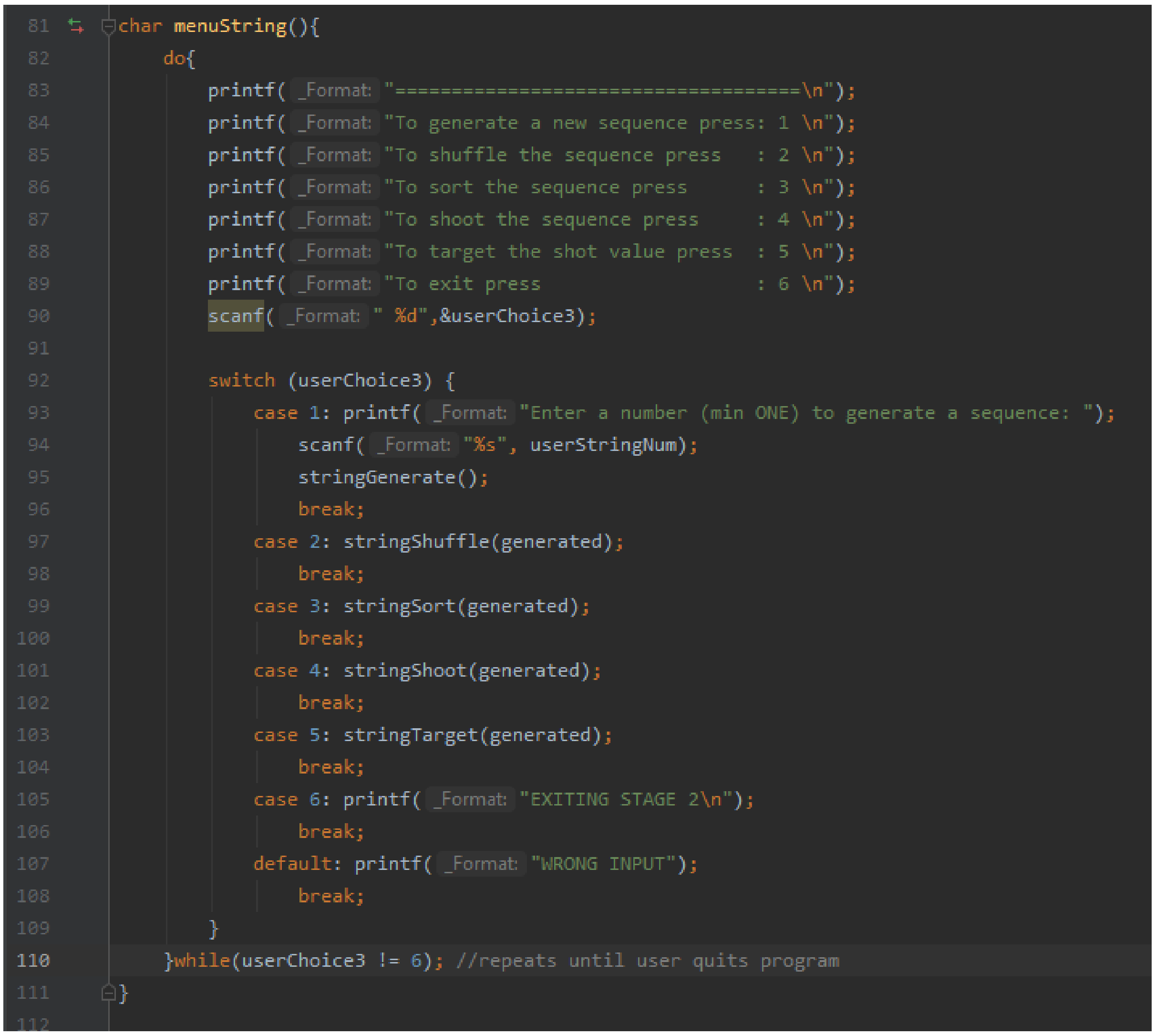


```c
int menu(){
    do{
        printf( _Format: "GREETINGS\n");
        printf( _Format: "====================================\n");
        printf( _Format: "1. To use numbers press          :1\n");
        printf( _Format: "2. to use letters (string) press :2\n");
        printf( _Format: "3. To quit press                 :3\n");
        printf( _Format: "INPUT: ");
        scanf( _Format: " %d", &userChoice);


        switch (userChoice) {
            case 1: menuInt();
                break;

            case 2: menuString();
                break;

            case 3: printf( _Format: "BYE");
                break;

            default: printf( _Format: "WRONG INPUT\n");
                break;
        }
    }while(userChoice !=3); //repeats until user quits program

}
```

In menuInt() the program asks the user which of the completed functions he would like to use. If any of the options are called upon without first generating the sequence the values will be all given as 0s. If the user picks the 1st case (to generate the sequence), he is asked to enter the number with which he wishes to begin the sequence. The value is passed through the function generate(), where it will work using the user’s input as its starting point. The other cases all call their respective functions, but instead the array gen is passed into them. Once again, the same concept is applied to the String version of the menu.

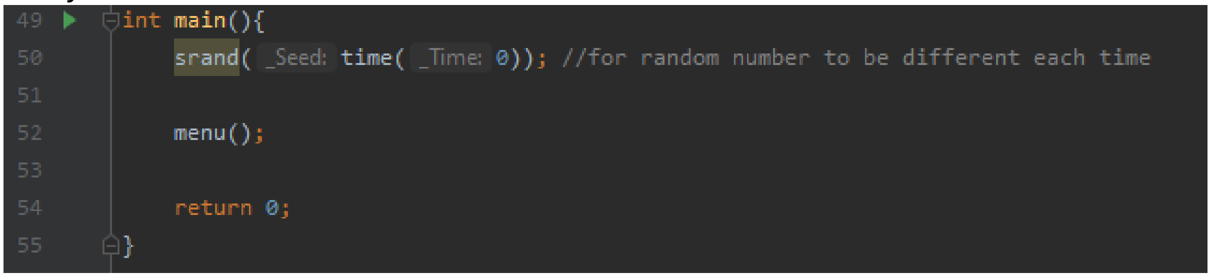


```c
int menuInt(){
    do{
        printf( _Format: "====================================\n");
        printf( _Format: "To generate a new sequence press: 1 \n");
        printf( _Format: "To shuffle the sequence press   : 2 \n");
        printf( _Format: "To sort the sequence press      : 3 \n");
        printf( _Format: "To shoot the sequence press     : 4 \n");
        printf( _Format: "To target the shot value press  : 5 \n");
        printf( _Format: "To exit press                   : 6 \n");
        scanf( _Format: " %d",&userChoice2);


        switch (userChoice2) {
            case 1: printf( _Format: "Enter a number (min 1) to generate a sequence: ");
                scanf( _Format: "%d", &userGen);
                generate(userGen);
                break;
            case 2:
                shuffle(gen);
                break;
            case 3:
                sort(gen);
                break;
            case 4:
                shoot(gen);
                break;
            case 5:
                target(gen);
                break;
            case 6:
                printf( _Format: "EXITING STAGE 1\n");
                break;
            default: printf( _Format: "WRONG INPUT");
                break;
        }
    }while(userChoice2 != 6); //repeats until user quits program
}
```



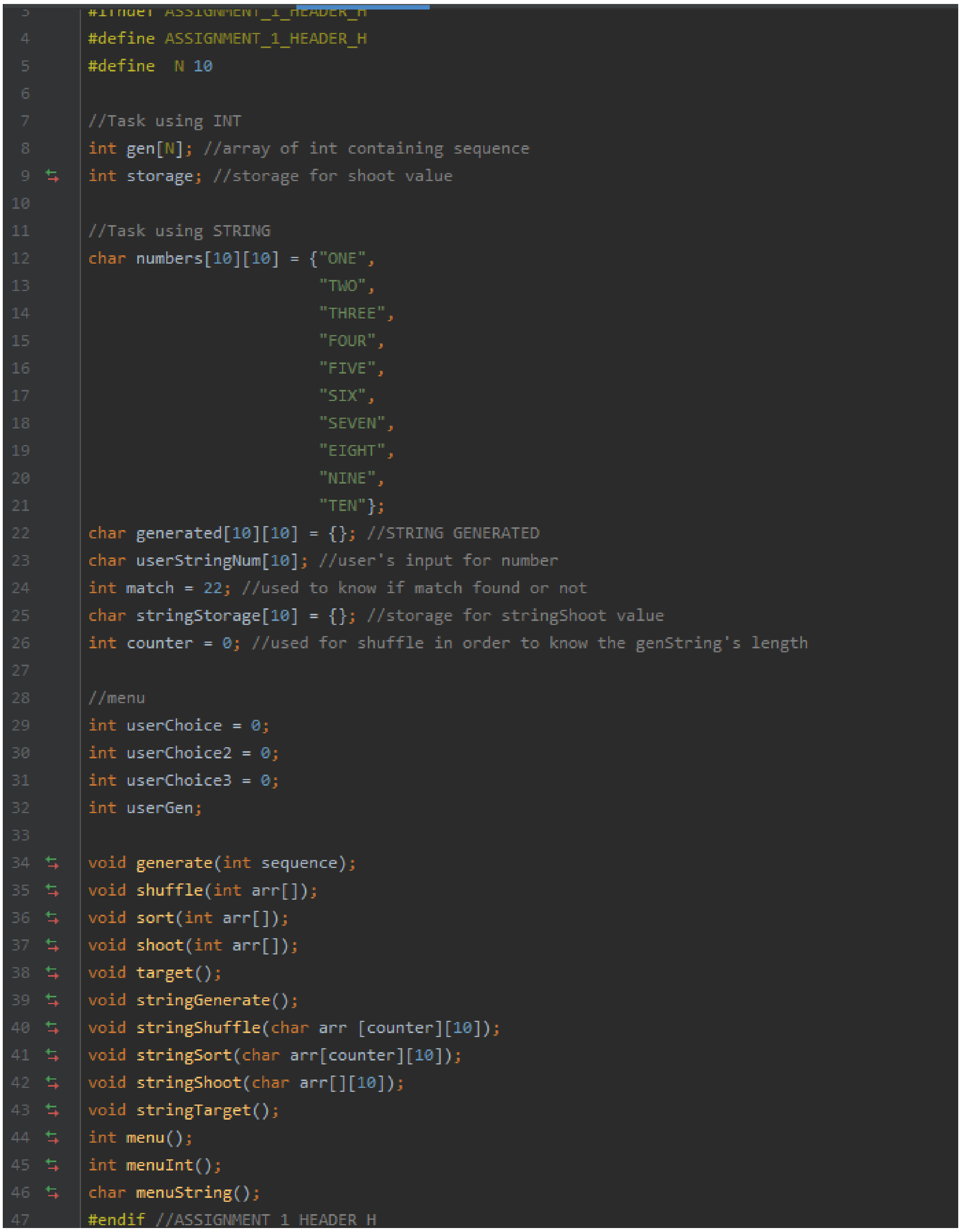
```c
char menuString(){
    do{
        printf( _Format: "====================================\n");
        printf( _Format: "To generate a new sequence press: 1 \n");
        printf( _Format: "To shuffle the sequence press   : 2 \n");
        printf( _Format: "To sort the sequence press      : 3 \n");
        printf( _Format: "To shoot the sequence press     : 4 \n");
        printf( _Format: "To target the shot value press  : 5 \n");
        printf( _Format: "To exit press                   : 6 \n");
        scanf( _Format: " %d",&userChoice3);

        switch (userChoice3) {
            case 1: printf( _Format: "Enter a number (min ONE) to generate a sequence: ");
                scanf( _Format: "%s", userStringNum);
                stringGenerate();
                break;
            case 2: stringShuffle(generated);
                break;
            case 3: stringSort(generated);
                break;
            case 4: stringShoot(generated);
                break;
            case 5: stringTarget(generated);
                break;
            case 6: printf( _Format: "EXITING STAGE 2\n");
                break;
            default: printf( _Format: "WRONG INPUT");
                break;
        }
    }while(userChoice3 != 6); //repeats until user quits program
}
```

In the main all we have is a function that calls menu and an srand that guarantees that every instance of random will have a different value



```c
int main(){
    srand( _Seed: time( _Time: 0)); //for random number to be different each time

    menu();

    return 0;
}
```

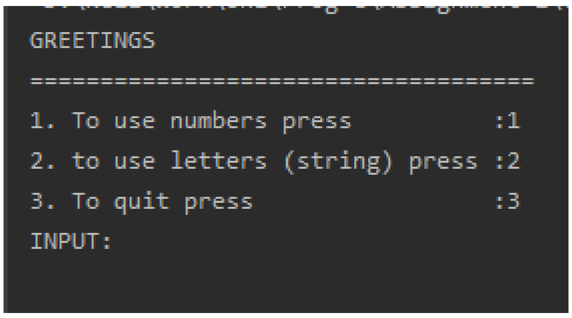
In the header file we have every declaration of the program, tucked away to make the main code more readable:



```c
#ifndef ASSIGNMENT_1_HEADER_H
#define ASSIGNMENT_1_HEADER_H
#define  N 10

//Task using INT
int gen[N]; //array of int containing sequence
int storage; //storage for shoot value

//Task using STRING
char numbers[10][10] = {"ONE",
                        "TWO",
                        "THREE",
                        "FOUR",
                        "FIVE",
                        "SIX",
                        "SEVEN",
                        "EIGHT",
                        "NINE",
                        "TEN"};
char generated[10][10] = {}; //STRING GENERATED
char userStringNum[10]; //user's input for number
int match = 22; //used to know if match found or not
char stringStorage[10] = {}; //storage for stringShoot value
int counter = 0; //used for shuffle in order to know the genString's length

//menu
int userChoice = 0;
int userChoice2 = 0;
int userChoice3 = 0;
int userGen;

void generate(int sequence);
void shuffle(int arr[]);
void sort(int arr[]);
void shoot(int arr[]);
void target();
void stringGenerate();
void stringShuffle(char arr [counter][10]);
void stringSort(char arr[counter][10]);
void stringShoot(char arr[][10]);
void stringTarget();
int menu();
int menuInt();
char menuString();
#endif //ASSIGNMENT_1_HEADER_H
```

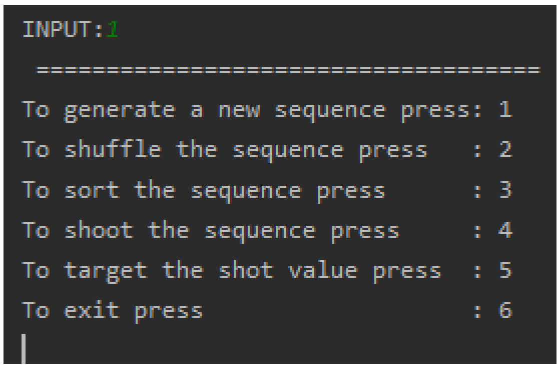
## Output listing

On start:



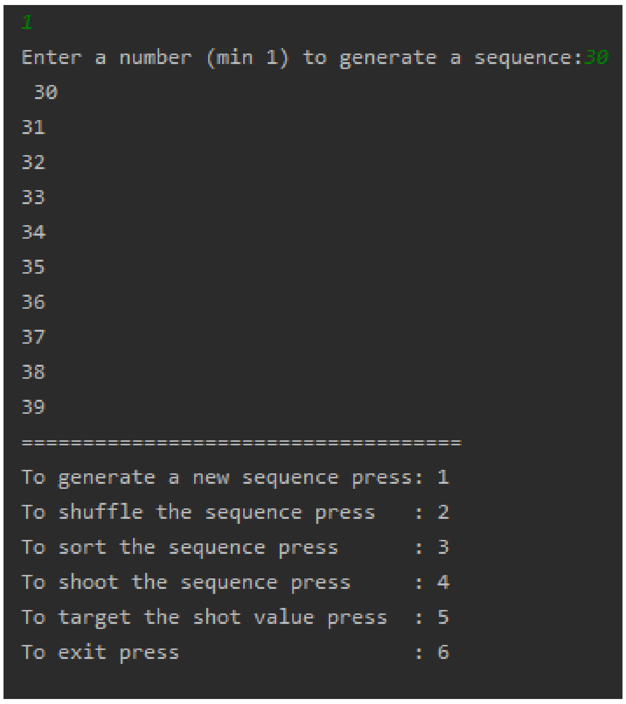
```
GREETINGS
====================================
1. To use numbers press          :1
2. to use letters (string) press :2
3. To quit press                 :3
INPUT:
```

## On selecting option 1:



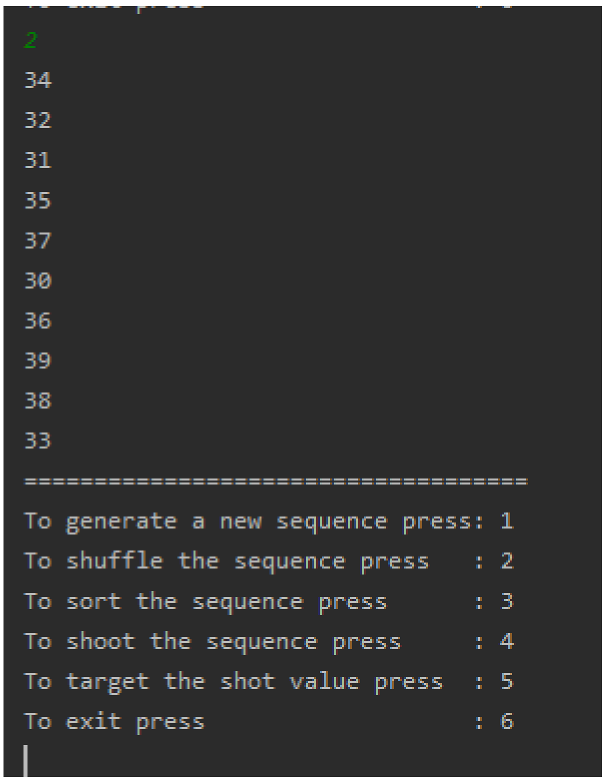
```
INPUT:1
 ====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

On generating new sequence:



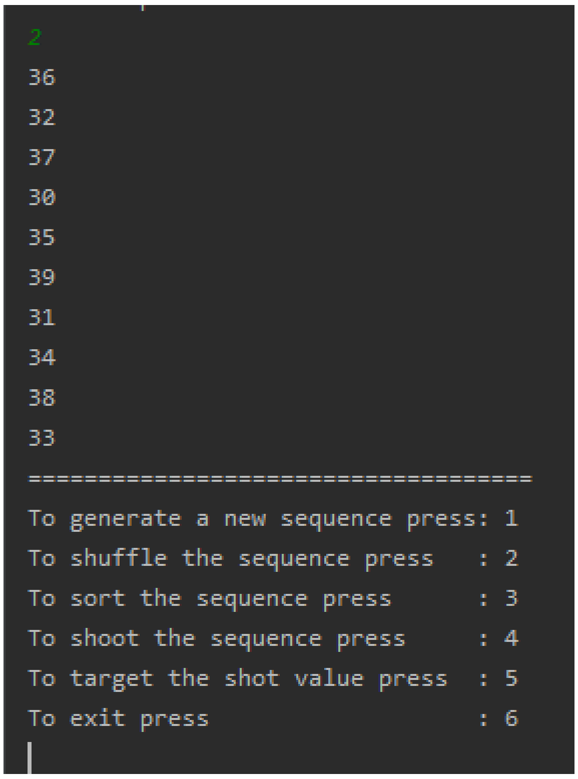
```
1
Enter a number (min 1) to generate a sequence:30
 30
31
32
33
34
35
36
37
38
39
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

On shuffling the array:



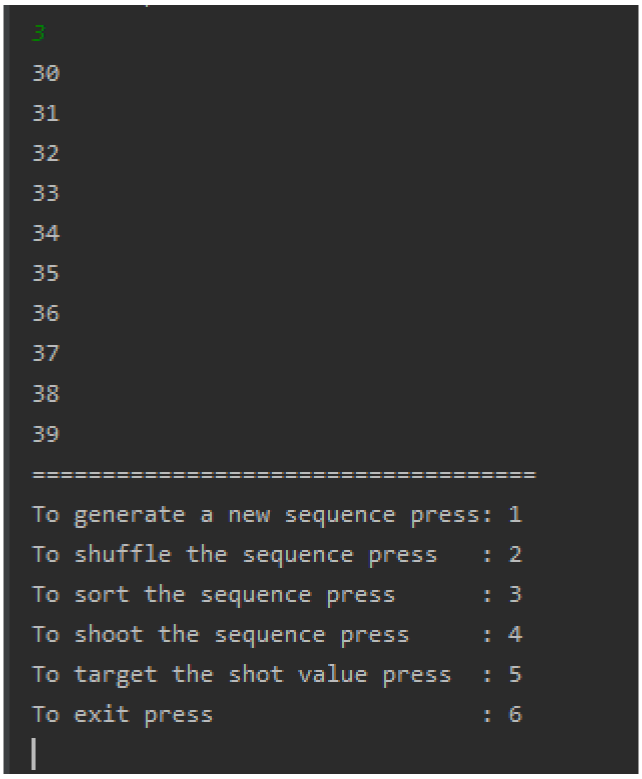
```
2
34
32
31
35
37
30
36
39
38
33
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

Notice the different answers when applying the generate function twice.

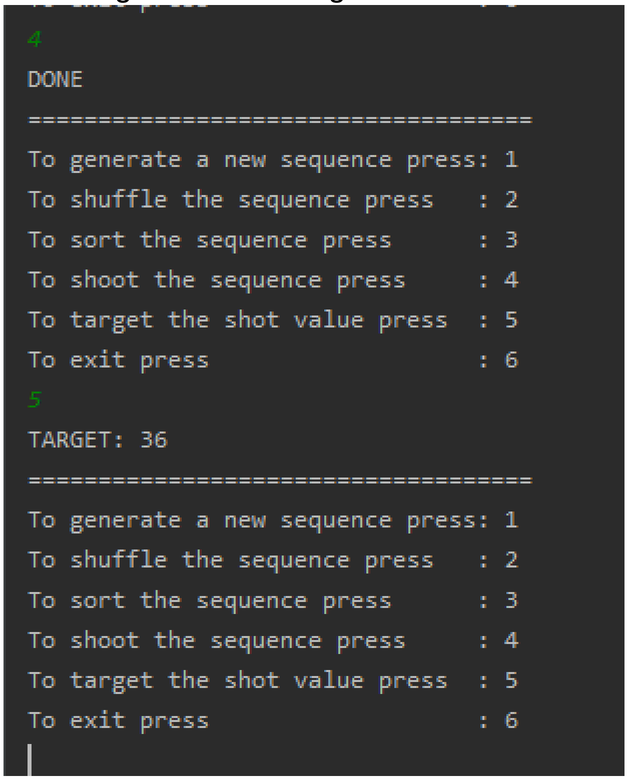


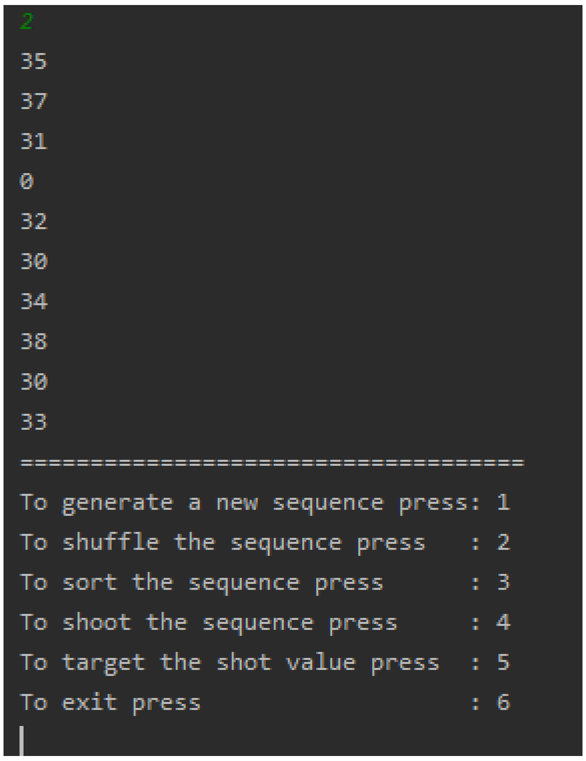
```
2
36
32
37
30
35
39
31
34
38
33
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

On sorting the sequence:


```
3
30
31
32
33
34
35
36
37
38
39
=====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

On using shoot and target:

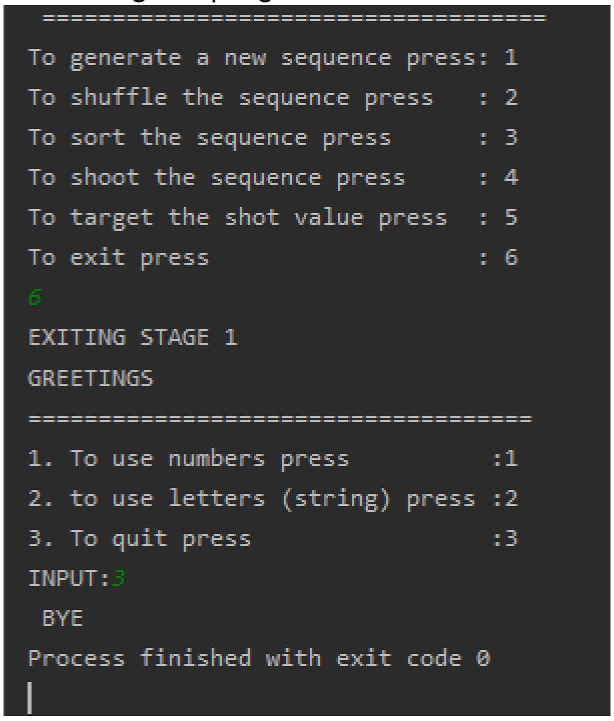

```
4
DONE
=====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
5
TARGET: 36
=====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```



```
2
35
37
31
0
32
30
34
38
30
33
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

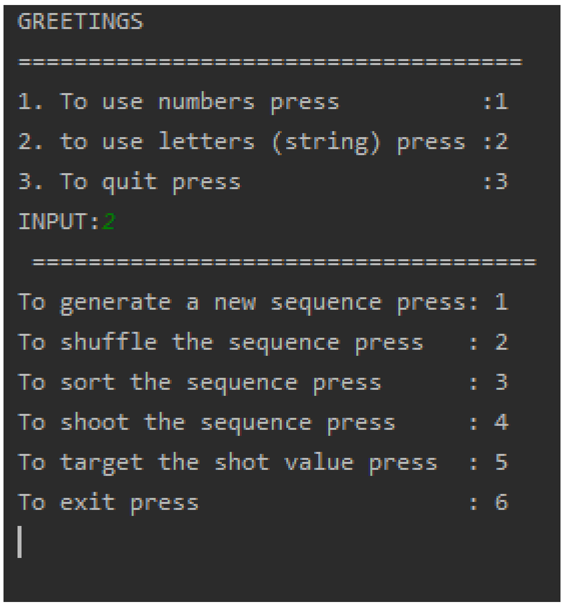
On applying the shuffle function we can see that the sequence has indeed been shot
through the presence of 0 instead of 36

On exiting the program:



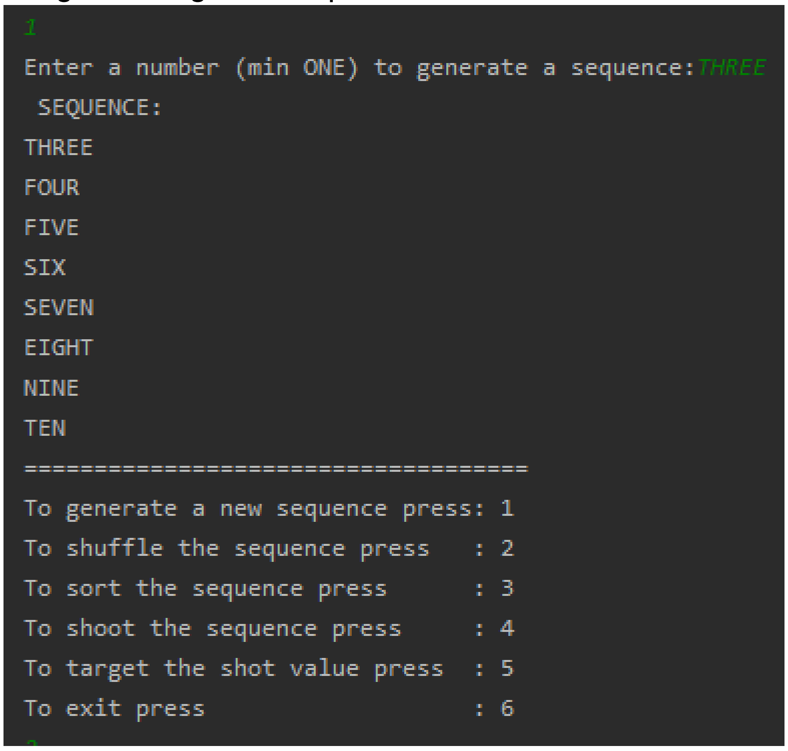
```
 ====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
6
EXITING STAGE 1
GREETINGS
====================================
1. To use numbers press          :1
2. to use letters (string) press :2
3. To quit press                 :3
INPUT:3
 BYE
Process finished with exit code 0
```

On selecting option 2:



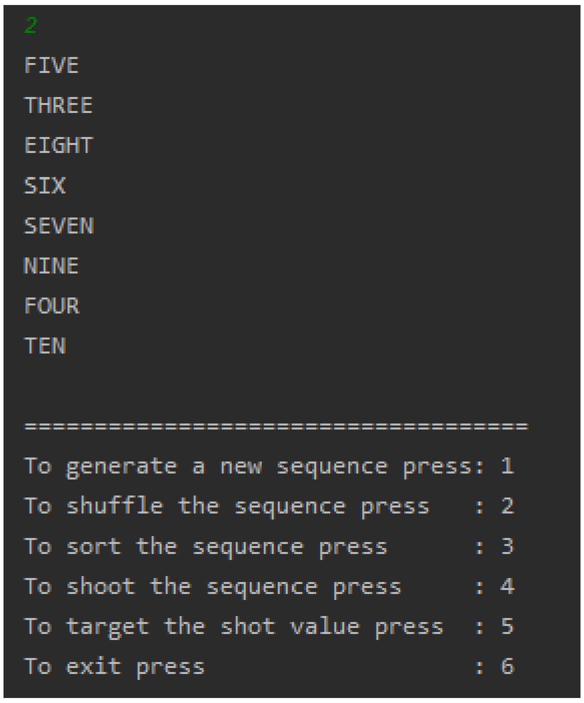
```
GREETINGS
====================================
1. To use numbers press          :1
2. to use letters (string) press :2
3. To quit press                 :3
INPUT:2
 ====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
|
```

On generating new sequence:



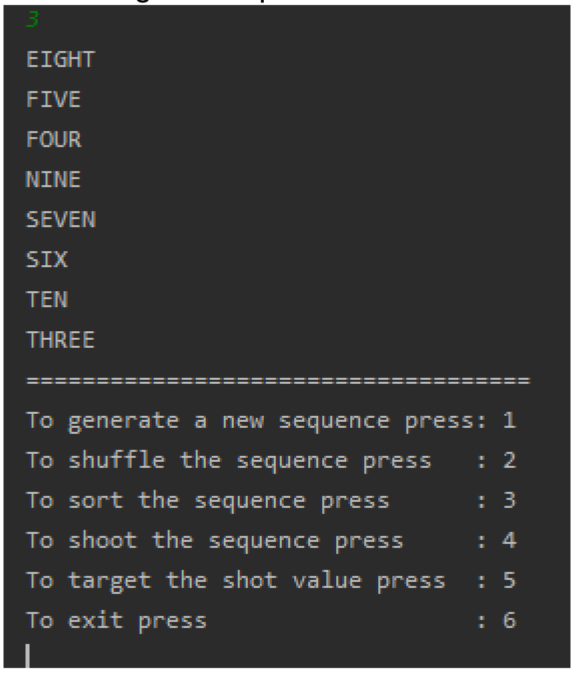
```
1
Enter a number (min ONE) to generate a sequence:THREE
 SEQUENCE:
THREE
FOUR
FIVE
SIX
SEVEN
EIGHT
NINE
TEN
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

On shuffling the sequence:



```
2
FIVE
THREE
EIGHT
SIX
SEVEN
NINE
FOUR
TEN

===================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

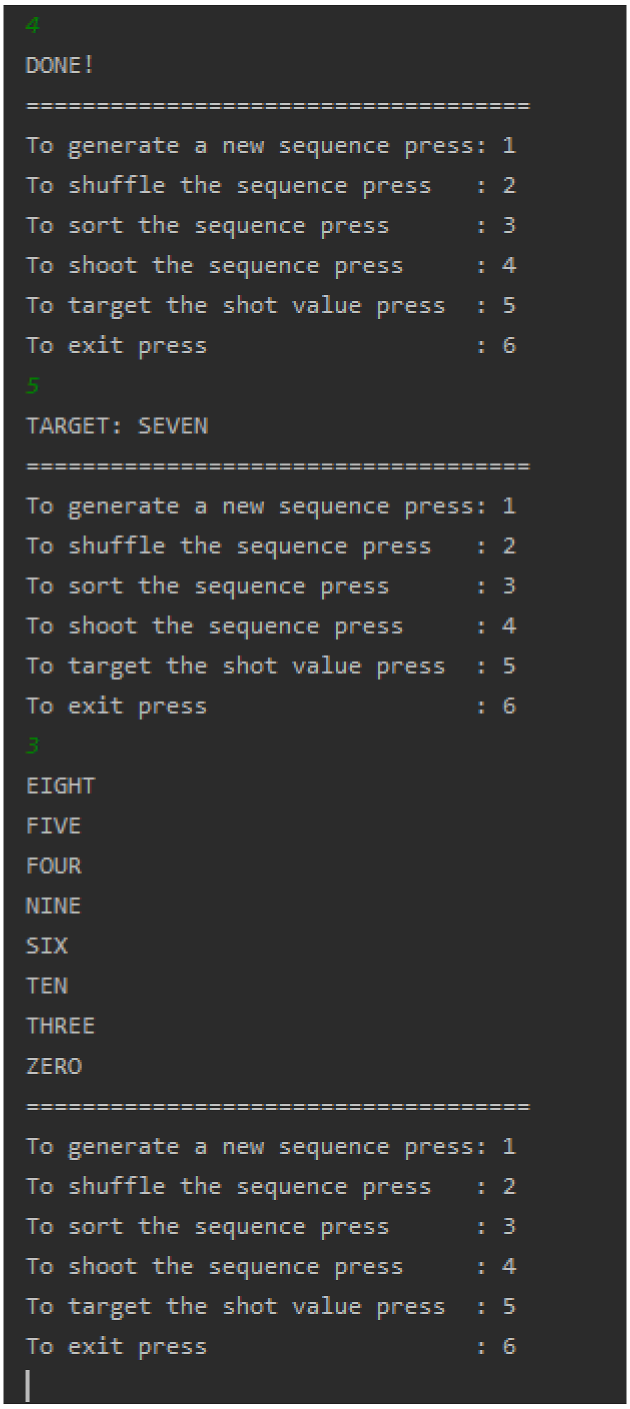
On sorting the sequence:



```
3
EIGHT
FIVE
FOUR
NINE
SEVEN
SIX
TEN
THREE
===================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

Sequence has been sorted alphabetically

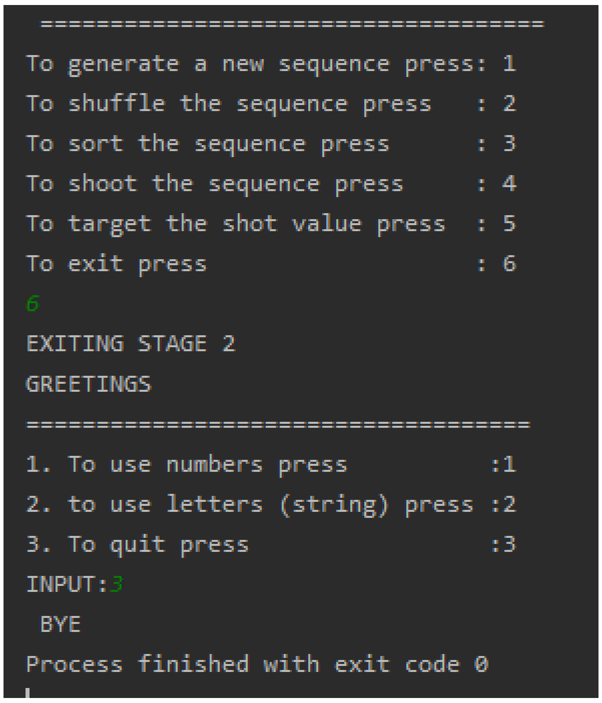
On shooting and targeting the sequence:



```
4
DONE!
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
5
TARGET: SEVEN
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
3
EIGHT
FIVE
FOUR
NINE
SIX
TEN
THREE
ZERO
====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
```

On applying the sort function we can see that the sequence has indeed been shot through the presence of "ZERO" instead of "SEVEN".

On exiting the program:



```
 ====================================
To generate a new sequence press: 1
To shuffle the sequence press   : 2
To sort the sequence press      : 3
To shoot the sequence press     : 4
To target the shot value press  : 5
To exit press                   : 6
6
EXITING STAGE 2
GREETINGS
====================================
1. To use numbers press          :1
2. to use letters (string) press :2
3. To quit press                 :3
INPUT:3
 BYE
Process finished with exit code 0
```

## References

Sorting arrays:
https://www.youtube.com/watch?v=6qiNJWw5aLI&list=LLVi2dA6rGVbt_kahhvIbyfw&index=10&t=609s

shuffling arrays:
https://www.youtube.com/watch?v=xH3VbMPFFec&list=LLVi2dA6rGVbt_kahhvIbyfw&index=9&t=0s