Problem 1.

Refer to the "Numerical Baseball Game" explained in pages 48 ~ 58 of the lecture material "Functions and Modular Programming".

Complete the **generate_target_number(void)** based on approach 2 explained in page 52 of the lecture material.

Approach 2: Randomly generate a number between 0 and 9, and specify it as the first digit. Again, randomly generate a number between 0 and 9, compare it with the first digit, and if it is different, assign it as the second digit, otherwise generate a new random number till it is different from the first digit. The third digit is determined in the same manner.

For the implementation of this function, use the given **generate_a_digit()** function.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/**
* @brief : generate and return a random digit in 0~9
* @return : a digit in 0~9
* @param   : none
*/
int generate_a_digit(void) {
  return (rand()%10);
}

/**
* @brief : generate a 3-digits target integer for the puzzle
* @return : return an integer with 3 different digits in 0~9
* @param : none
*/
int generate_target_number_approach2(void) {
  // Implement here

  return 123;
}

```

c

Problem 2.

Complete the function **get_match_result()** for the numerical baseball game explained in page 55 of the lecture material "07. Functions and Modular Programming".

This function compares the guessed number with the target and returns the match result for the numerical baseball game explained in page 48.

```c
#include <stdio.h>
/**
* @brief : compare the guessed number with the target and return the match results
* @return : return 2-digits number, 1st digit stands for the number of strikes and 2nd digit stands for the number of balls. For example 11 means 1 strike 1 ball
* @param  : target - 3-digits target integer
          guessed - 3-digits guessed integer
*/
int get_match_result(int target, int guessed) {
  int n_strike=0, n_ball=0;

  // Implement here

  return (n_strike*10 + n_ball);
}

```

Problem 3.

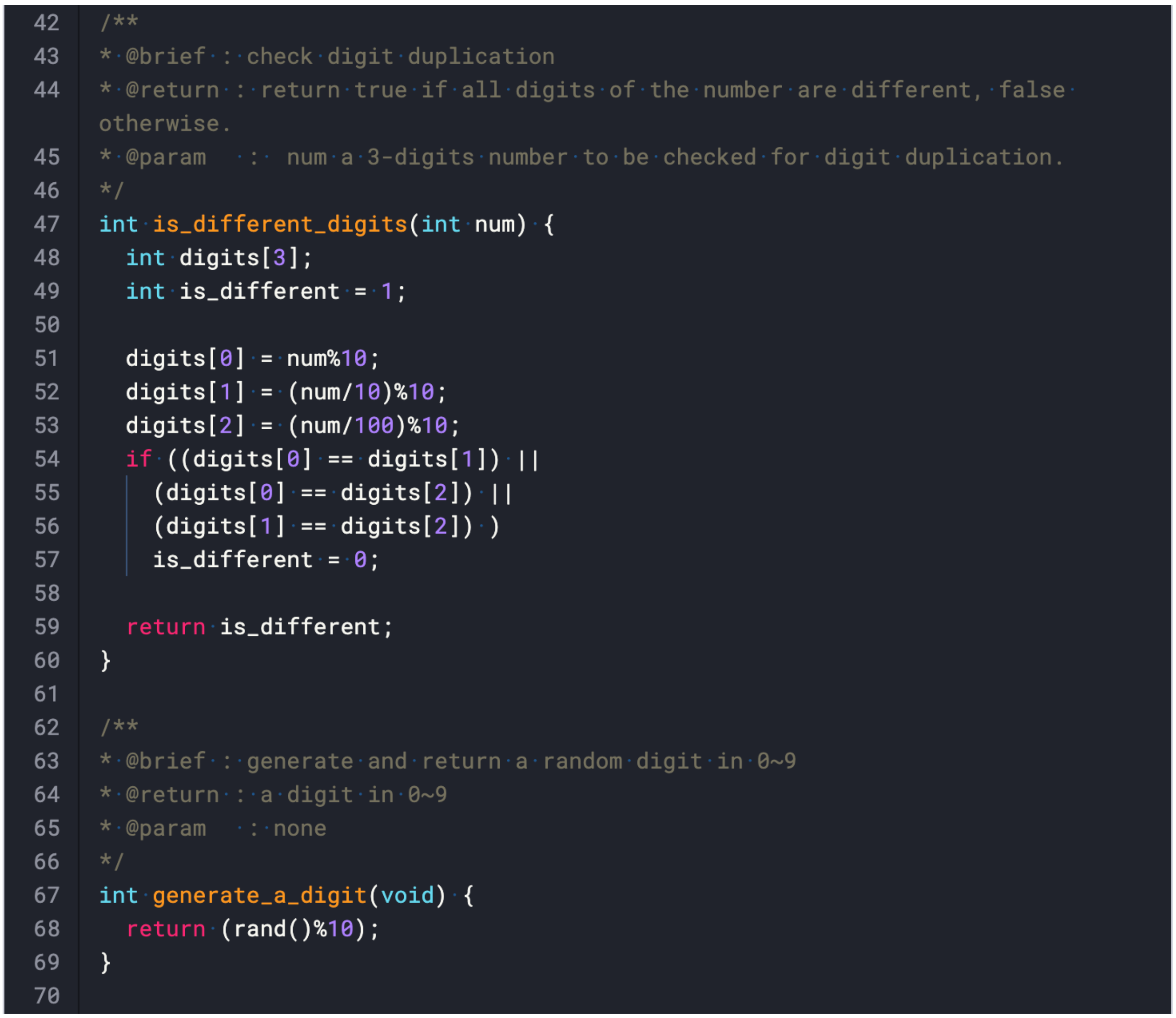
Complete a numerical baseball game program by including the 2 functions, generate_target_number() and get_match_result(), you wrote in the previous problems.

Execute the program and try to solve the puzzle. (The time limit is 120 seconds.)

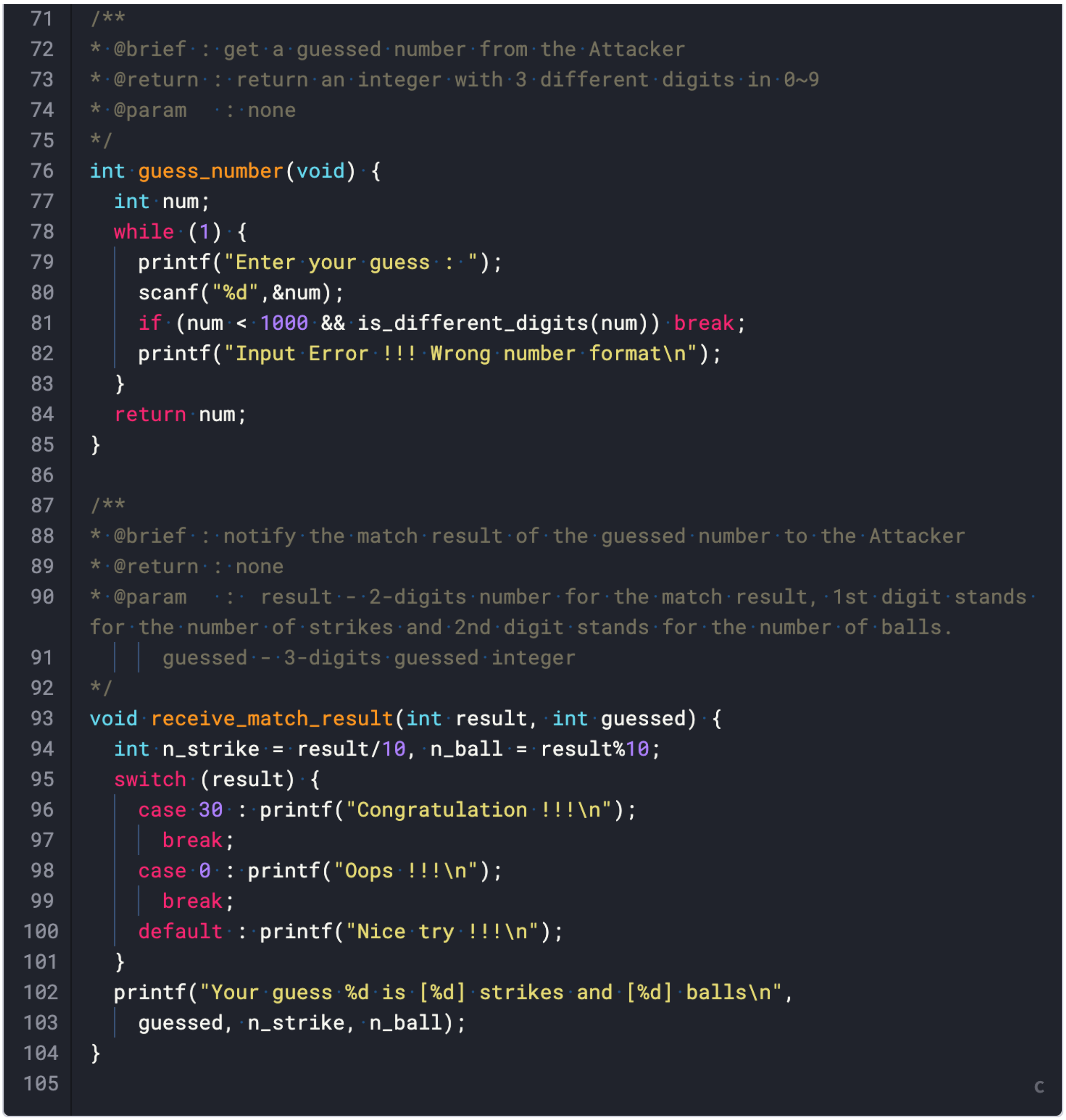
Your T.A will check your code and your execution result (you should show a case that you win) for scoring.

(이전 문제에서 작성한 generate_target_number() 와 get_match_result()를 주어진 코드에 추가하여 숫자야구게임을 완성하라. 프로그램을 완성한 후 실행하고 퍼즐을 풀어보라. 시간 제한은 120초이다. 문제를 맞춘 경우 조교에게 채점을 요청하면 조교가 당신의 코드와 실행결과를 확인하고 점수를 부여할 것이다.)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define ATTEMPT_LIMIT 9

// Function Prototypes
int generate_target_number(void);
int guess_number(void);
int get_match_result(int target, int guessed);
void receive_match_result(int result, int guessed);

int is_different_digits(int num);
int generate_a_digit(void);

// Implement generate_target_number()

// Implement get_match_result()


int main(void) {
  int n_attempt = 0;
  int target_num, guessed_num, match_result;

  setvbuf(stdout,NULL,_IONBF, 0);

  srand(time(0));
  target_num = generate_target_number();
  do {
    printf("Attempt [%d/%d] ",++n_attempt, ATTEMPT_LIMIT);
    guessed_num = guess_number();
    match_result = get_match_result(target_num, guessed_num);
    receive_match_result(match_result, guessed_num);
  } while (n_attempt <= ATTEMPT_LIMIT && match_result != 30);

  if (match_result == 30)
    printf("Congratulation!!! You Win\n");
  else
    printf("Nice Try!!! But You Lose, The target number is %d\n", target_num);
  return 0;
}
```



```c
/**
* @brief : check digit duplication
* @return : return true if all digits of the number are different, false otherwise.
* @param   :  num a 3-digits number to be checked for digit duplication.
*/
int is_different_digits(int num) {
  int digits[3];
  int is_different = 1;

  digits[0] = num%10;
  digits[1] = (num/10)%10;
  digits[2] = (num/100)%10;
  if ((digits[0] == digits[1]) ||
    (digits[0] == digits[2]) ||
    (digits[1] == digits[2]) )
    is_different = 0;

  return is_different;
}

/**
* @brief : generate and return a random digit in 0~9
* @return : a digit in 0~9
* @param   : none
*/
int generate_a_digit(void) {
  return (rand()%10);
}

```



```c
/**
* @brief : get a guessed number from the Attacker
* @return : return an integer with 3 different digits in 0~9
* @param   : none
*/
int guess_number(void) {
  int num;
  while (1) {
    printf("Enter your guess : ");
    scanf("%d",&num);
    if (num < 1000 && is_different_digits(num)) break;
    printf("Input Error !!! Wrong number format\n");
  }
  return num;
}

/**
* @brief : notify the match result of the guessed number to the Attacker
* @return : none
* @param   :  result - 2-digits number for the match result, 1st digit stands for the number of strikes and 2nd digit stands for the number of balls.
      guessed - 3-digits guessed integer
*/
void receive_match_result(int result, int guessed) {
  int n_strike = result/10, n_ball = result%10;
  switch (result) {
    case 30 : printf("Congratulation !!!\n");
      break;
    case 0 : printf("Oops !!!\n");
      break;
    default : printf("Nice try !!!\n");
  }
  printf("Your guess %d is [%d] strikes and [%d] balls\n",
    guessed, n_strike, n_ball);
}
```

Problem 4.

In 1949 the Indian mathematician D.R. Kaprekar discovered a class of numbers called self-numbers. For any positive integer *n*, define *d(n)* to be *n* plus the sum of the digits of *n*. (The d stands for Digitaddition, a term coined by Kaprekar.)

For example, *d(75)* = 75 + 7 + 5 = 87. Given any positive integer *n* as a starting point, you can construct the infinite increasing sequence of integers *n, d(n), d(d(n)), d(d(d(n))),* ... .

For example, if you start with 33, the next number is 33 + 3 + 3 = 39, the next is 39 + 3 + 9 = 51, the next is 51 + 5 + 1 = 57, and so you generate the sequence

33, 39, 51, 57, 69, 84, 96, 111, 114, 120, 123, 129, 141, ...

The number *n* is called a generator of *d(n)*. In the sequence above, 33 is a generator of 39, 39 is a generator of 51, 51 is a generator of 57, and so on. Some numbers have more than one generator: for example, 101 has two generators, 91 and 100. **A number with no generators is a self-number**. There are 13 self-numbers less than 100: 1, 3, 5, 7, 9, 20, 31, 42, 53, 64, 75, 86, and 97.

Write a program to output all positive self-numbers less than 10000 in increasing order, one per line.

You can add new functions and variables.

```c
#include <stdio.h>
#define MAX_NUMBER  10000

// Implement here

int main(void) {

  return 0;
}

```