2022암호경진대회 3번 문제

저희는 setup() 함수 안의 반복문과 Cracking() 함수를 수정하여 고속화하였습니다.

아래 1, 2 번은 두 함수를 수정한 이유와 상세 수정 내용에 대한 설명입니다.

최종 결과는 3 번에 사진으로 첨부해 두었습니다.

1. setup() 반복문 수정 이유

먼저, init_int을 0부터 99999999까지 1씩 증가시키며 찾으므로 , 시간이 가장 오래 걸릴 암호는 99999999라는 것을 알 수 있습니다. 아래 이어지는 답안에서는 암호가 99999999일 경우를 최대라 표현하였습니다.

문제 코드를 그대로 실행할 경우, 앞 4자리, 뒤 4자리를 각각 Func1, Func2 함수에서 변환한 후 다시 모두 합쳐 8자리를 한꺼번에 비교합니다. 이렇게 비교하면 각각의 경우의 수인 10000을 두 번 곱하여 최대 100000000번의 변환 후 정답과의 비교과정을 거치게 됩니다.

Cracking시 Func1, Func2함수가 각각 앞 4자리, 뒤 4자리만을 이용하는 것에서 착안하여 앞 4자리만 먼저 변환하고 정답과 일치할 때 뒤 4자리를 찾는 코드를 구현해보았습니다. 이 경우, 0~9999 사이 중 정답과 일치하는 앞 4자리를 찾는데 최대 10000번 Func1함수를 실행하고, 뒤 4자리도 같은 원리로 Func2함수를 최대 10000번 실행합니다. 이 때 앞 4자리가 일치해야만 뒤 4자리 변환함수를 실행하므로 최대 실행 횟수는 각각의 경우의 수를 더한 20000번입니다.

위의 방법으로 반복문을 수정할 시 원래 코드대로 실행할 때보다 검사횟수가 최대 1/5000로 줄어든다는 것을 알 수 있습니다.

a. 수정 전

```
time1 = millis();


 init_int = 0;
 for (i = 0; I < 99999999; i++) {
```

```
  check = Cracking(init_int, password, output, answer_bench);
  if (check) {
   Serial.print("Answer is ");
   Serial.println(init_int);
   break;
  }

  for (j = 0; j < 8; j++) { output[j] = 0; }
  init_int++;
}
time2 = millis();
```

b. 수정 후

```
time1 = millis();

init_int = 0;
for (i = 0; i < 10000; i++){
 check = Cracking(i, password, output, answer_bench);
 if (check){
   break; /*정답과 일치하는 앞 4 자리 찾은 후 반복문 빠져나감*/
 }
 for (j = 0; j < 4; j++){
   output[j] = 0;
 }
 init_int += 1;
```

```
}


for (i = 0; i < 10000; i++){ /*뒤 4 자리 탐색*/
  check = Cracking(i, password, output, &answer_bench[4]);
  if (check){
   Serial.print("Answer is ");
   Serial.println(init_int);
   break;
  }
  for (j = 0; j < 4; j++){ output[j] = 0; }
}


time2 = millis();
```

2. Cracking 함수 설명

a. 레퍼런스 코드

```
u8 Cracking(u32 init_int, u8* password, u8* output, u8* answer){
 u8 check;
 int_to_char(init_int, password);  /*숫자를 8 글자로 문자화*/
 Func1(password, output); /*앞의 4 글자 암호화*/
 Func2(&password[4], &output[4]); /*뒤의 4 글자 암호화*/
 check=Matching(output, answer);  /*암호화 된 8 글자 answer 과 동일여부 확인*/
 return check;
}
```

b. 바꾼 코드

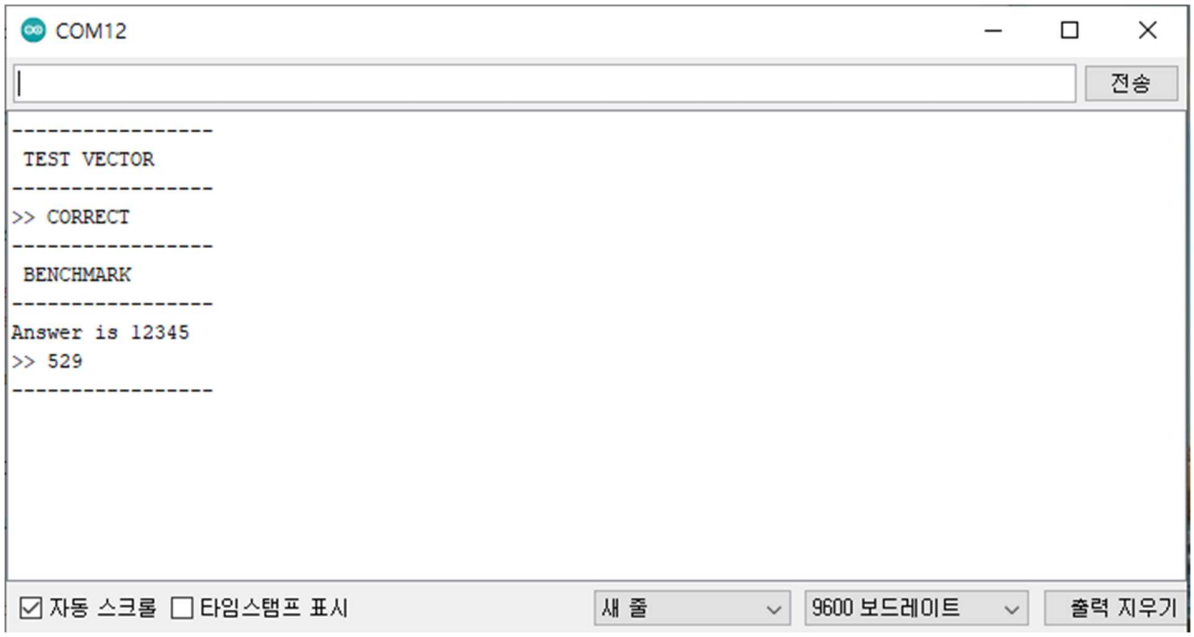
Func1, Func2의 4글자만 사용해서 암호화를 한다는 특성을 이용해서 setup()의 반복문을 두 개로 나누고, 각 반복문에서 4글자씩만을 비교하면 충분합니다. 반면에 테스트 벡터에서 사용될 때는 8글자 모두를 비교해야 합니다.

결론적으로 Cracking 함수가 호출되는 경우는 테스트 벡터에서 1번, 반복문에서 2번, 총 3번입니다. 이 서로 다른 3번의 실행을 함수가 구분할 수 있게 하기 위해 함수 내부에 static int 변수 3개를 선언하고, 그 변수들의 값에 따라 서로 다른 코드가 실행되게 했습니다.

```
u8 Cracking(u32 init_int, u8 *password, u8 *output, u8 *answer){


  static u8 check1 = 0;  /*테스트 벡터 실행 결과 저장*/
  static u8 check2 = 0;  /*첫 번째 반복문 결과 저장*/
  static u8 check3 = 0;  /*두 번째 반복문 결과 저장*/
```

```
  if (check1 == 0){            /*테스트 벡터 미통과 => 테스트 벡터 실행*/

    int_to_char(init_int, password);    /*8 글자 모두 문자화*/
    Func1(password, output);            /*앞의 4 글자 암호화*/
    Func2(&password[4], &output[4]);    /*뒤의 4 글자 암호화*/
    check1 = Matching(output, answer);  /*8 글자 전부 비교*/
    return check1;

  } else if (check2 == 0){      /*테스트 벡터 통과, 첫번째 반복문 미통과 => 첫번째 반복문*/
    int_to_char_4bit(init_int, password);   /*4 글자만 문자화*/
    Func1(password, output);                /*4 글자만 암호화*/
    check2 = Matching_4bit(output, answer); /*4 글자만 비교*/
    return check2;

  } else if (check3 == 0){      /*테스트 벡터 및 첫번째 반복문 통과, 두번째 반복문 미통과 => 두번째
반복문 실행*/
    int_to_char_4bit(init_int, password);   /*4 글자만 문자화*/
    Func2(password, output);                /*4 글자만 암호화*/
    check3 = Matching_4bit(output, answer); /*4 글자만 비교*/
    return check3;

  }
}
```

3. 실행 결과


COM12
전송

TEST VECTOR

>> CORRECT

BENCHMARK

Answer is 12345
>> 529

자동 스크롤
타임스탬프 표시
새 줄
9600 보드레이트
출력 지우기