# C 프로그래밍

# 9 주차 과제

{

제출: 2022년 10월 31일

학과: 지능미디어공학과

학번: 20221085

이름: 김기홍

**03**

로그인 시에 아이디를 검사하는 함수 int check()를 작성해서 테스트하라. check()가 한번 호출될 때마다 비밀번호를 질문하고 일치 여부를 0과 1으로 반환한다. 비밀번호는 숫자 1234로 고정되어 있다고 가정한다. check()가 3번이상 호출되고 아이디가 일치하지 않으면 check()는 "로그인 시도 횟수 초과" 메시지를 출력한다. check() 함수 안에 정적 변수를 선언하여 사용해보자.

```c
//로그인 횟수제한
#include <stdio.h>

#define Pass 1
#define Fail 0

#define pswd 1234        //고정된 비밀번호

unsigned __int8 count_over = 0;  //로그인(시도) 자격

int check(void);

int main(void)
{
        while (count_over==0)
        {
                if (check() == Pass)
                        break;
        }

        return 0;
}

int check(void)
{
        static unsigned __int8 count = 0;//정적 부호없는 8비트정수 카운트
        unsigned int input;

        if (count >= 3){
                count_over = 1;
                printf("로그인 시도 횟수 초과\n");
                return Fail;
        }
        count++;//로그인 시도 횟수 1 증가

        printf("비밀번호: ");
        scanf_s("%ud", &input);

        if (pswd == input){      //비밀번호 일치 여부 확인
                printf("로그인 성공\n");
                return Pass;
        }
        else
                return Fail;
}
```

**전처리기, 전역변수**

**Pass, Fail:**
아이디를 검사하는 함수 check()는 비밀번호 일치 여부에 따라서 1(일치)과 0(불일치)을 반환한다. 1과 0은 추상적이고 혼동이 올 수 있는 숫자이므로 비밀번호 일치(1)를 반환할때는 기호상수 Pass를, 불일치(0)를 반환할때는 기호상수 Fail을 반환하도록 한다.

**Pswd:**
비밀번호는 1234로 고정되어 있다. 즉, 런타임 중에 비밀번호가 바뀌지 않으므로 Pswd라는 기호상수로 정의해주었고, check()함수에서도 사용해야 하기 때문에 const가 아닌 #define을 사용했다.

**count_over:**
후술할 count지역 변수에 따라 로그인 시도 자격을 결정할 변수이다. 로그인 시도 자격을 check()함수 내에서 정하고 main()함수에서도 이를 사용해서 while문을 탈출해야하기 때문에 전역변수로 선언했다. 0 혹은 1의 값만 가져도 되기 때문에 크기가 가장 작은 자료형인 __int8을 사용했다.

**check() 함수**

**변수**

**static unsigned __int8 count:**
count변수는 로그인 시도 횟수를 저장하는 역할을 한다. 함수의 작동이 끝나더라도 로그인 시도 횟수는 저장돼있어야 하므로 static을 사용해서 정적변수로 만들어준다.
카운트는 음의 값을 가질 필요가 없으므로 unsigned를 사용해서 부호를 없애준다.
그리고 로그인 시도는 3회가 최대이므로 count의 값은 8비트 정수 범위인 255를 넘지 않아도 된다.
그러므로 __int8을 사용해서 변수의 자료형을 8비트(1 byte) 정수로 선언해 주었다. __int8자료형 대신 char자료형을 사용해도 똑같이 작동한다.

**unsigned int input:**
input변수는 사용자에게 비밀번호를 입력받는 역할을 한다. 비밀번호는 음수가 될 일이 없기 때문에 unsigned를 사용해 부호를 없앴다.

**로그인 시도 횟수**

로그인 시도 횟수가 3회를 넘으면 더 이상 비밀번호를 비교하지 않아야 한다. 로그인 시도 횟수를 저장할 count변수가 3이상의 값을 가지고 있다면 로그인 시도 자격을 없애고, 로그인 시도 횟수가 초과되었다는 문구를 출력, 로그인 실패를 뜻하는 기호상수 Fail을 반환한다.
또한 count변수의 값을 증가시키는 count++ 는 로그인 시도 횟수를 확인하는 코드 뒤에 작성하였는데,

이는 count변수의 오버플로우를 방지하기 위함이다.
만약 count++가 상단에 위치해 있고, count변수가 3의 값을 가지고 있을 때 check()함수가 253번 호출된다면 count변수의 값이 0으로 오버플로우 되고 사용자가 비밀번호를 다시 비교할 수 있게된다.

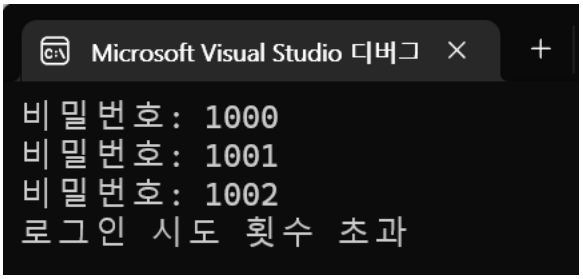
3+253 = 256 = 255(0b11111111)+1 = 0(0b00000000)

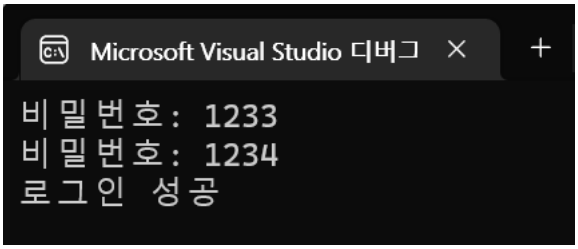
**비밀번호 비교**

사용자에게 입력받은 비밀번호를 프로그램에 저장된 비밀번호와 비교한다. 일치한다면 '로그인 성공' 문구를 출력하고, 로그인 성공을 뜻하는 기호상수 Pass를 반환한다. 일치하지 않는다면 로그인 실패를 뜻하는 기호상수 Fail을 반환한다.

**main() 함수**

main()함수는 로그인 자격이 없어질 때까지 check()함수를 호출하는 동작을 한다. check()함수를 호출했을 때 로그인 성공을 뜻하는 기호상수 Pass를 반환받으면 break으로 반복문을 탈출한다.
로그인 자격이 사라진다면 while문의 비교결과가 거짓이 되면서 반복문을 탈출한다.



**04**

본문에서 설명한 바와 같이 정적 변수는 초기화를 딱 한 번만 수행하는 경우에도 사용된다. 난수를 생성하여 반환하는 함수 get_random()을 작성하여 테스트하라. get_random()이 처음으로 호출되는 경우에는 srand()를 호출하여서 난수의 시드를 초기화하고 그렇지 않으면 단순히 rand()를 호출하여 난수를 반환한다.

```c
//랜덤함수 초기화
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int get_random(void);
int main(void)
{

        for (int i = 0; i < 3; i++)
                printf("%d\n", get_random());

        return 0;
}
int get_random(void)
{
        static __int8 inited = 0;
        if (inited == 0){
                srand((unsigned)time(NULL));
                printf("초기화 실행\n");
                inited = 1;
        }
        return rand();
}
```

**get_random()함수**

**변수**

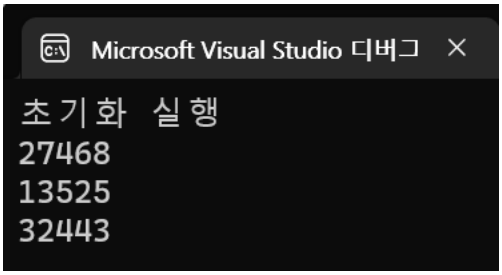
**static __int8 inited:**
inited변수는 rand()함수의 시드가 초기화 된 적 있는지를 저장한다.
0혹은 1의 값만 가져도 되므로 8비트 정수형인 __int8자료형을 사용한다.

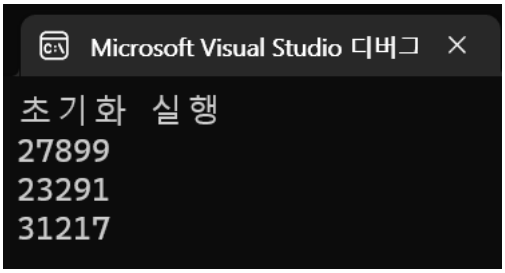
**초기화**

inited변수의 값이 0이라면, 즉 rand함수의 시드가 한번도 초기화되지 않은 경우에 srand()함수의 인자로 time()함수의 반환값을 전달해서 시드를 초기화한다. 그리고 '초기화 실행' 이라는 문구를 출력한 다음, inited변수의 값을 1으로 바꾼다.

**반환**

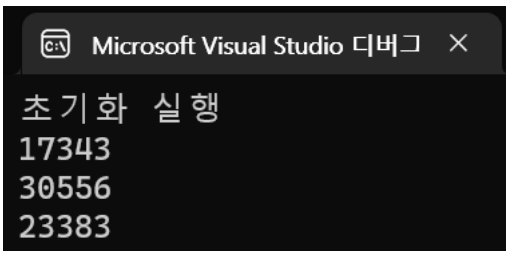
rand() 함수를 호출해서 그 값을 반환한다.

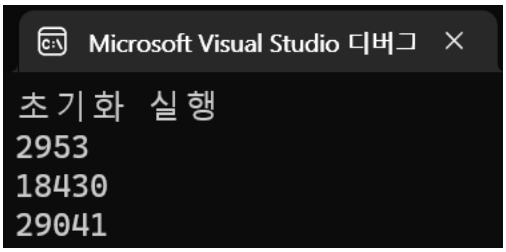



프로그램을 실행할 때 마다 첫번째 값이 크게 변하지 않는 것을 볼 수 있었는데, 이는 time()함수의 반환값에 큰 수를 곱함으로써 분포를 크게할 수 있었다.

```
if (inited == 0){
        srand((unsigned)time(NULL)<<16);
        printf("초기화 실행\n");
        inited = 1;
}
```



**06**

순환기법을 이용하여 지수값을 계산하는 함수 power(int base, int power_raised)를 작성하고 테스트해보자. power(2, 3)가 호출되면 2^3을 계산하여 반환한다.

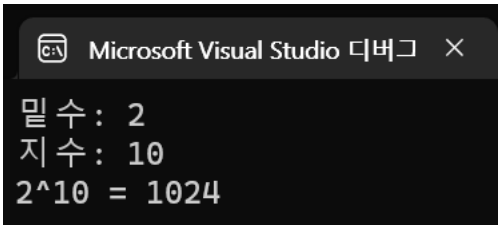
```c
//순환함수 제곱
#include <stdio.h>

int power(int base,int power_raised);

int main(void)
{
	int x;
	int y;

	printf("밑수: ");
	scanf_s("%d", &x);

	printf("지수: ");
	scanf_s("%d", &y);

	printf("%d^%d = %d\n", x, y, power(x, y));

	return 0;
}

int power(int base, int power_raised)
{
	if (power_raised == 1)
		return base;
	return base * power(base, power_raised - 1);
}
```

**power(int base, int power_raised) 함수**

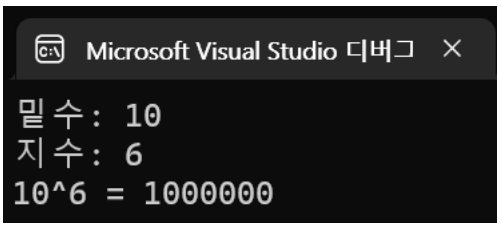
power()함수가 2^3의 지수값을 계산하는 과정은 다음과 같이 풀어 쓸 수 있다.

power(2,3) → (2 * power(2,2)) → (2 * (2 * power(2,1))) → (2 * (2 * (2)))

결과적으로 power(2,3) 함수에서 8이 반환된다.


Microsoft Visual Studio 디버그
밑수: 2
지수: 10
2^10 = 1024


Microsoft Visual Studio 디버그
밑수: 3
지수: 4
3^4 = 81


Microsoft Visual Studio 디버그
밑수: 10
지수: 6
10^6 = 1000000

**08**

주어진 정수가 몇 개의 자리수를 가지고 있는지를 계산하는 프로그램을 순환을 이용하여 작성해보자. 예를 들어서 12345의 경우에는 5가 출력된다.

```c
//자리수 세기
#include <stdio.h>

int countdigit(int n);

int main(void)
{
	int n;

	printf("정수를 입력하시오: ");
	scanf_s("%d", &n);

	printf("자리수의 개수: %d\n", countdigit(n));

	return 0;
}

int countdigit(int n)
{
	if (n < 10)
		return 1;
	return 1 + countdigit(n / 10);
}
```

| 깊이 | 매개변수(countdigit(n)) | 반환값 (return) |
|---|---|---|
| 1 | 12345 | return 1 + countdigit(1234) |
| 2 | 1234 | return 1 + countdigit(123) |
| 3 | 123 | return 1 + countdigit(12) |
| 4 | 12 | return 1 + countdigit(1) |
| 5 | 1 | return 1 |

**12**

순환호출을 이용하여 피보나치 수열을 계산해 보자.

```c
//피보나치

#include <stdio.h>
#include <time.h>

int fib(int n);

int main()
{
	clock_t t = clock();

	for (int i = 0; i < 40; i++)
		printf("fib(%d) = %d\n", i, fib(i));

	printf("소요시간: %d ms\n", clock() - t);

	return 0;
}

int fib(int n)
{
	if (n <= 1)
		return n;
	else
		return fib(n - 2) + fib(n - 1);
}
```

n번째 피보나치 수는 n-2번과 n-1번 수의 합과 같다.

이를 식으로 표현하면 fib(n)=fib(n-2)+fib(n-1)이다.

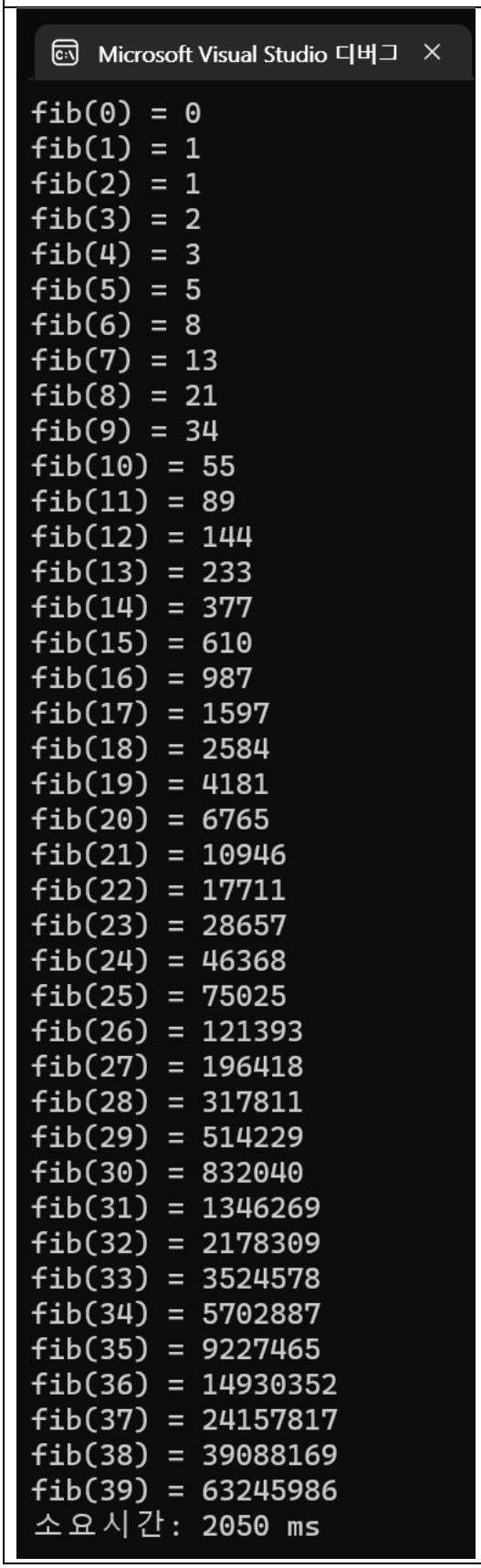
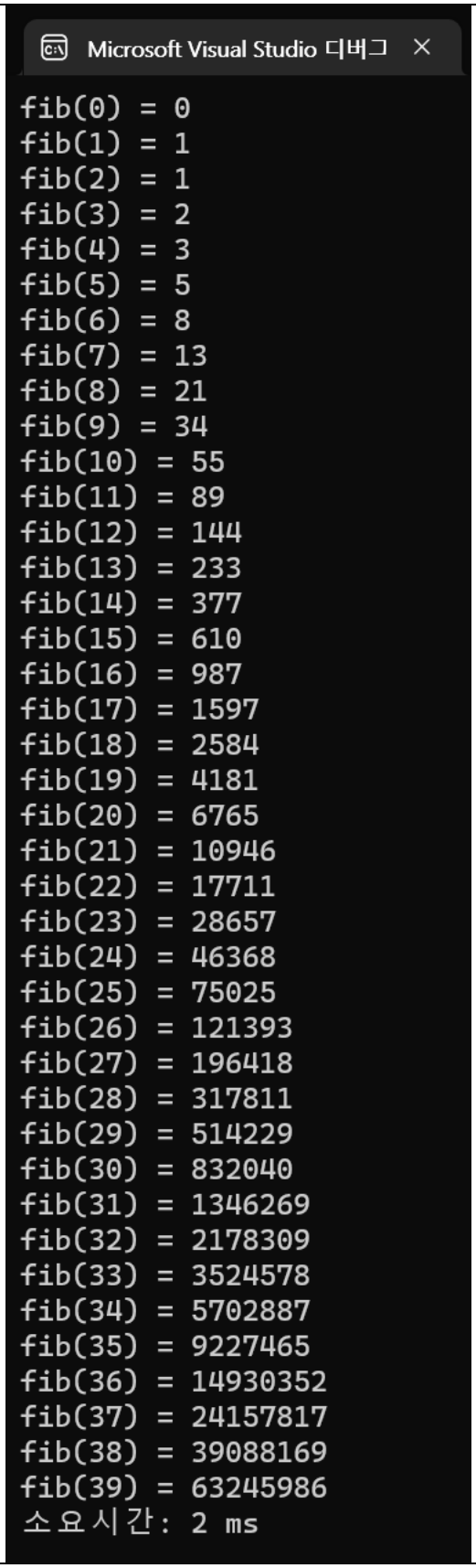
위의 코드에서는 fib(n)을 호출하면 fib(n-2)와 fib(n-1)의 값을 더해서 반환한다.
fib(n)는 fib(n-2), fib(n-1)를
fib(n-2)는 fib(n-4), fib(n-3)를,
fib(n-1)은 fib(n-3), fib(n-2)를 호출한다.
이 호출은 n이 1이나 0이 될 때까지 진행한다.
호출되는 fib()함수를 보면 매개변수가 똑같은 함수가 호출되는 경우가 많다. 이러한 중복호출은 n의 값이 커질수록 피보나치 수를 구하는 시간이 오래걸리게 만든다.

이를 Dynamic Programming(Memoization) 방식으로 최적화해보았다.

```c
//피보나치 DP

#include <stdio.h>
#include <time.h>

int fib(int n);

int m[40] = { 0 };

int main()
{
	clock_t t = clock();

	for (int i = 0; i < 40; i++)
		printf("fib(%d) = %d\n", i, fib(i));

	printf("소요시간: %d ms\n", clock() - t);

	return 0;
}

int fib(int n)
{
	if (n <= 1)
		return n;

	if (m[n])
		return m[n];
	else
		return m[n] = fib(n - 2) + fib(n - 1);
}
```

배열에 fib()의 매개변수에 따른 반환 값(n번째 피보나치 수)을 저장하고, fib()함수를 호출할 때 배열에 저장된 값을 반환하도록 한다.

만약 fib(4), fib(5), fib(6)이 순서대로 호출된다면, fib(4)가 구해지는 과정에서 0번~4번까지의 피보나치 수가 배열에 저장되고 fib (5)는 배열의 3번째 칸과 4번째 칸의 값을 더해서 반환하면 된다. 이는 fib(6)에서도 동일하게 4번째 칸과 5번째 칸의 값을 더해서 반환하면 되므로, 중복호출이 매우 적어진다.

| DP 미사용 | DP사용 |
| --- | --- |
|  |  |

Microsoft Visual Studio 디버그
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
fib(10) = 55
fib(11) = 89
fib(12) = 144
fib(13) = 233
fib(14) = 377
fib(15) = 610
fib(16) = 987
fib(17) = 1597
fib(18) = 2584
fib(19) = 4181
fib(20) = 6765
fib(21) = 10946
fib(22) = 17711
fib(23) = 28657
fib(24) = 46368
fib(25) = 75025
fib(26) = 121393
fib(27) = 196418
fib(28) = 317811
fib(29) = 514229
fib(30) = 832040
fib(31) = 1346269
fib(32) = 2178309
fib(33) = 3524578
fib(34) = 5702887
fib(35) = 9227465
fib(36) = 14930352
fib(37) = 24157817
fib(38) = 39088169
fib(39) = 63245986
소요시간: 2050 ms

Microsoft Visual Studio 디버그
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
fib(10) = 55
fib(11) = 89
fib(12) = 144
fib(13) = 233
fib(14) = 377
fib(15) = 610
fib(16) = 987
fib(17) = 1597
fib(18) = 2584
fib(19) = 4181
fib(20) = 6765
fib(21) = 10946
fib(22) = 17711
fib(23) = 28657
fib(24) = 46368
fib(25) = 75025
fib(26) = 121393
fib(27) = 196418
fib(28) = 317811
fib(29) = 514229
fib(30) = 832040
fib(31) = 1346269
fib(32) = 2178309
fib(33) = 3524578
fib(34) = 5702887
fib(35) = 9227465
fib(36) = 14930352
fib(37) = 24157817
fib(38) = 39088169
fib(39) = 63245986
소요시간: 2 ms