1. 함수 square(int)를 정의하여 입력받은 정수의 제곱 수를 출력하는 프로그램을 작성하시오.

입력 예시 1

```
3
```

출력 예시 1

```
제곱할 정수 입력 : 3
3의 제곱 : 9
```

입력 예시 2

```
8
```

출력 예시 2

```
제곱할 정수 입력 : 8
8의 제곱 : 64
```

2. plus(int)에는 입력받은 정수에 10을 더하여 반환하고, minus(int)는 10을 뺀 값을 반환하는 함수를 정의하고, 이 값을 출력하는 프로그램을 작성하시오.

입력 예시 1

```
20
```

출력 예시 1

```
정수 입력 : 20
10 더한 값 : 30
10 뺀 값 : 10
```

입력 예시 2

```
60
```

출력 예시 2

```
정수 입력 : 60
10 더한 값 : 70
10 뺀 값 : 50
```

5. 사칙연산을 하는 함수를 선언하고, 두 개의 정수와 하나의 연산자를 입력받아 사칙연산의 결과를 출력하는 프로그램을 작성하시오. (+, -, *, / 이외의 연산은 0으로 출력)

입력 예시 1

```
5 9 -
```

출력 예시 1

```
두 정수 입력 : 5 9
연산자 입력 : -
5 - 9 = -4
```

입력 예시 2

```
8 6 /
```

출력 예시 2

```
두 정수 입력 : 8 6
연산자 입력 : /
8 / 6 = 1
```

입력 예시 3

```
5 2 =
```

출력 예시 3

```
두 정수 입력 : 5 2
연산자 입력 : =
5 = 2 = 0
```

2. 두 개의 정수를 입력받아(n, a) n의 a 제곱을 구하는 함수를 정의(pow)하고 이를 출력하는 프로그램을 작성하시오.

입력 예시 1

```
2 3
```

출력 예시 1

```
두 정수를 입력하세요 : 2 3
2의 3 제곱 : 8
```

입력 예시 2

```
2 0
```

출력 예시 2

```
두 정수를 입력하세요 : 2 0
2의 0 제곱 : 1
```

입력 예시 3

```
5 1
```

출력 예시 3

```
두 정수를 입력하세요 : 5 1
5의 1 제곱 : 5
```

3. 자연수(n)를 입력받아 n!(팩토리얼)을 구하는 프로그램을 작성하시오. 예를 들어 5! = 5 * 4 * 3 * 2 * 1 = 120이 출력된다.

입력 예시 1

```
5
```

출력 예시 1

```
자연수를 입력하세요 : 5
5! = 120
```

입력 예시 2

```
1
```

출력 예시 2

```
자연수를 입력하세요 : 1
1! = 1
```

입력 예시 3

```
8
```

출력 예시 3

```
자연수를 입력하세요 : 8
8! = 40320
```

4. 자연수(n)를 입력받아 약수의 개수를 구하는 프로그램을 작성하시오.

입력 예시 1

```
8
```

출력 예시 1

```
자연수를 입력하세요 : 8
8의 약수 개수 : 4
```

입력 예시 2

```
20
```

출력 예시 2

```
자연수를 입력하세요 : 20
20의 약수 개수 : 6
```

입력 예시 3

```
99
```

출력 예시 3

```
자연수를 입력하세요 : 99
99의 약수 개수 : 6
```

■ 변수의 존재기간과 접근범위 1 : 지역변수

- 변수의 선언위치와 함수는 깊은 관계가 있음.
- 변수는 선언되는 위치에 따라서 '**전역변수**'와 '**지역변수**'로 나뉨.
- 메모리상에 존재하는 기간과 변수에 접근할 수 있는 범위에 차이점을 보임.
- 함수 내에서만 존재 및 접근 가능한 지역 변수(Local Variable)
- 지역변수 예제

```c
#include <stdio.h>
int SimpleFuncOne(int);
int SimpleFuncTwo(int, int);
int main()
{
    int num1 = 10;  // 이후부터 main 함수 내에서 num1 변수 접근 가능
    int num2 = 12;  // 이후부터 main 함수 내에서 num2 변수 접근 가능
    SimpleFuncOne(num1);
    SimpleFuncTwo(num1, num2);
    printf("main num1 : %d, num2 : %d\n", num1, num2);
    return 0;       // main의 num이 유효한 마지막 문장
}
int SimpleFuncOne(int num)
{
    num++;
    printf("SimpleFuncOne num : %d\n", num);
    return 0;
}
int SimpleFuncTwo(int num1, int num2)
{
    int num3 = 30;
    num1++, num2--;

    { // 함수내에서 중괄호로 새로운 영역을 생성
        int num1 = 40;
        int num2 = 50;
        printf("num1 : %d\n", num1);
        printf("num2 : %d\n", num2);
        printf("num3 : %d\n", num3);
    }

    printf("SimpleFuncTwo num1 : %d, num2 : %d, num3 : %d\n", num1, num2, num3);
    return 0;
}
```

실행결과

- 함수를 정의할 때 선언하는 매개변수도 지역변수의 일종이다.
- 매개변수는 선언된 함수 내에서만 접근이 가능함
- 매개변수는 선언된 함수가 반환을 하면, 지역변수와 마찬가지로 소멸됨.

■ 변수의 존재기간과 접근범위 2 : 전역변수

- 프로그램의 시작과 동시에 메모리 공간에 할당되어 종료시까지 존재
- 별도의 값으로 초기화 하지 않으면 0으로 초기화 됨.
- 프로그램 전체 영역 어디서든 접근이 가능함.
- 전역변수 예제(1)

```c
#include <stdio.h>
void Add(int val);
int num;    // 전역변수는 기본 0으로 초기화됨
int main()
{
    printf("num : %d\n", num);
    Add(3);
    printf("num : %d\n", num);
    num++;      // 전역변수 num의 1 증가
    printf("num : %d\n", num);
    return 0;
}
void Add(int val)
{
    num += val;
}
```

실행결과

- 전역변수 예제(2)-전역변수와 동일한 이름의 지역변수가 선언된 경우

```c
#include <stdio.h>
int Add(int val);
int num = 1;    // 전역변수
int main()
{
    int num = 5;    // 지역변수
    printf("num : %d\n", Add(3));
    printf("num : %d\n", num+9);    // num은 지역변수? 전역변수?
    return 0;
}
int Add(int val)
{
    int num = 9;
    num += val;
    return num;
}
```

실행결과

```
num :
num :
```

- 해당 지역내에서는 전역변수가 가려지고, 지역변수로 접근이 이뤄짐.

변수	지역변수	전역변수
존재 기간	함수 실행되고 함수가 끝날 때까지	프로그램이 끝날 때까지
접근 범위	함수 안에서만 접근 가능	프로그램 어디서든 접근 가능

■ 변수의 존재기간과 접근범위 3 : static 변수

- 지역변수의 static을 선언하면 이는 전역변수의 성격을 지님.
- 선언된 함수 내에서만 접근이 가능(지역변수 특성)
- 초기화는 딱 1회, 이후 프로그램 종료시까지 메모리 공간에 존재(전역변수 특성)
- static 변수 예제(1)

```c
#include <stdio.h>
void SimpleFunc();
int main()
{
    int i;
    for (i = 1; i <= 5; i++)
    {
        SimpleFunc();
    }
    return 0;
}
void SimpleFunc()
{
    // 접근성은 SimpleFunc() 내부로 제한된 지역변수이나
    // 정의는 이 함수가 여러번 호출되더라도 단 한번만 적용됨.
    static int num1 = 0; // 초기화 하지 않으면 0 초기화
    int num2 = 0;        // 초기화 하지 않으면 임의의 값으로 초기화
    num1++, num2++;
    printf("static : %d, local : %d\n", num1, num2);
}
```

실행결과

```
static : 1, local : 1
static : 2, local : 1
static : 3, local : 1
static : 4, local : 1
static : 5, local : 1
```

- static 선언한 변수는 프로그램 시작과 동시에 메모리 할당 및 초기화 됨.
- 프로그램이 종료될때까지 메모리 공간이 남아있음.
- 접근 범위가 함수 내부로 제한되어 있음!

- static 변수 연습(1)

다음은 프로그램 사용자가 입력하는 값을 누적하여 그 합계를 출력하는 예제이다.

```c
#include <stdio.h>
int total = 0;
int addToTotal(int num);
int main()
{
    int num, i;
    for (i = 0; i < 3; i++) {
        printf("입력%d : ", i + 1);
        scanf("%d", &num);
        printf("\t누적 : %d\n", addToTotal(num));
    }
}
```

```c
int addToTotal(int num)
{
    total += num;
    return total;
}
```

위의 예제에서 함수 addToTotal에서의 사용을 목적으로 전역변수 int total을 선언하였는데, 이를 static 변수로 대체해보자. 단, static 변수로의 대체과정에서 int main()함수의 변경은 없어야 하며 실행결과도 동일해야 한다.