



C프로그래밍

Lecture 10. C언어의 핵심! 함수!

동덕여자대학교
데이터사이언스 전공
권 범

목차

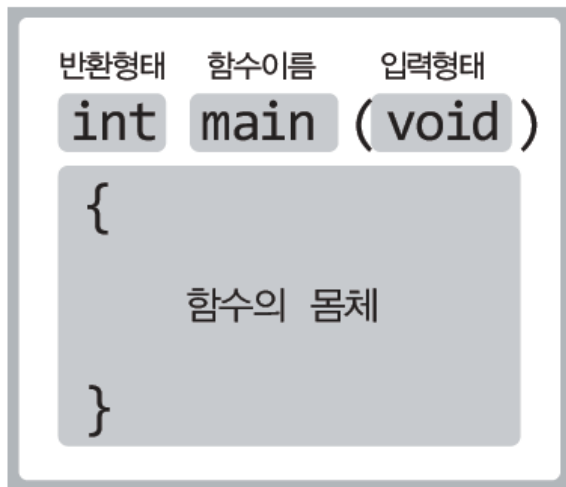
- ❖ 01. 함수를 정의하고 선언하기
- ❖ 02. 변수의 존재 기간과 접근 범위: 지역변수
- ❖ 03. 전역변수, static 변수, register 변수
- ❖ 04. 재귀함수에 대한 이해
- ❖ 05. 연습 문제

01. 함수를 정의하고 선언하기

- 02. 변수의 존재 기간과 접근 범위: 지역변수
- 03. 전역변수, static 변수, register 변수
- 04. 재귀함수에 대한 이해
- 05. 연습 문제

01. 함수를 정의하고 선언하기

❖ 함수를 만드는 이유 (1/3)



✓ C언어의 핵심은 함수입니다.

✓ 함수를 잘 정의하는 것도 중요하고, 잘 정의된 함수를 가져 다 쓰는 것도 중요합니다.

✓ 함수를 잘 구성하는 프로그래머는 실력 있는 프로그래머이고, 함수를 적절히 구성하지 못하는 프로그래머는 아무리 문법적으로 탄탄해도 그 실력을 인정받지 못합니다.

01. 함수를 정의하고 선언하기

❖ 함수를 만드는 이유 (2/3)

✓ 프로그래밍을 한다는 것은 복잡한 문제를 해결하는 것과 같습니다.



✓ 문제를 main이라는 하나의 함수 안에서 해결하다 보면 복잡해질 뿐만 아니라 그만큼 더 어려워집니다.



✓ 구현에 필요한 기능들이 어떤 것이 있는지 분석해서 각각 독립된 함수로 구현하고, 이들을 하나의 프로그램으로 완성시켜 가야지만 좋은 프로그램을 작성할 수 있습니다.

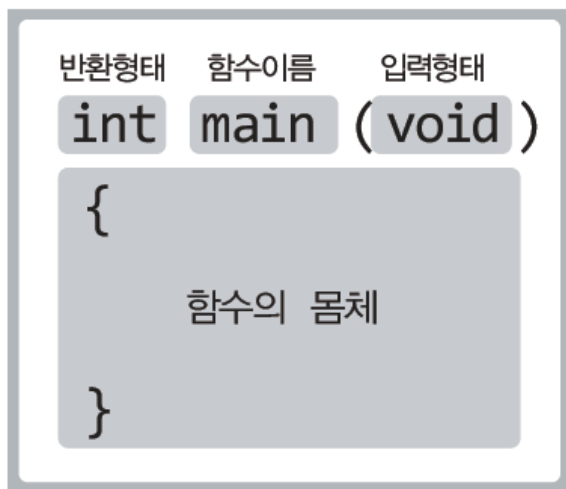


✓ 함수를 나눈다는 것은 복잡한 문제를 작은 문제로 나눠가면서 해결한다는 의미입니다.

01. 함수를 정의하고 선언하기

❖ 함수를 만드는 이유 (3/3)

- main 함수를 포함하여 함수의 크기는 작을수록 좋습니다.
- 무조건 작다고 좋은 것은 아니지만, 불필요하게 큰 함수가 만들어지지 않도록 주의해야 합니다.



- ✓ 하나의 함수는 하나의 일만 담당하도록 디자인 되어야 합니다.
- ✓ 물론 하나의 일이라는 것은 매우 주관적인 기준입니다.
- ✓ 그러나 이러한 주관적 기준 역시 프로그래밍에 대한 경험이 쌓이면 매우 명확한 기준이 됩니다.

01. 함수를 정의하고 선언하기

❖ 함수의 입력과 출력: printf 함수도 반환을 합니다.

```
1  /* return_printf.c */
2  #include <stdio.h>
3  int main(void)
4  {
5      int num1, num2;
6
7      num1 = printf("12345\n");
8      num2 = printf("I love you.\n");
9
10     printf("%d %d\n", num1, num2);
11
12     return 0;
13 }
```

12345
I love you.
6 12

- ✓ printf 함수도 사실상 값을 반환합니다.
- ✓ 반환값이 필요 없어서 반환되는 값을 저장하지 않았을 뿐입니다.
- ✓ printf 함수는 출력된 문자열의 길이를 반환합니다.

- ✓ 함수가 값을 반환하면 반환된 값이 함수의 호출문을 대체한다고 생각하면 됩니다.
- ✓ 예를 들어 아래의 printf 함수 호출문이 6을 반환한다면,

```
num1 = printf("12345\n");
```

- ✓ 함수의 호출 결과는 다음과 같이 되어 대입 연산이 진행됩니다.

```
num1 = 6;
```

01. 함수를 정의하고 선언하기

❖ 함수의 구분

- 전달인자와 반환 값의 유무에 따라 4가지 유형으로 구분할 수 있습니다.

◆ 유형 1: 전달인자 있고, 반환 값 있다! 전달인자(○), 반환 값(○)

◆ 유형 2: 전달인자 있고, 반환 값 없다! 전달인자(○), 반환 값(X)

◆ 유형 3: 전달인자 없고, 반환 값 있다! 전달인자(X), 반환 값(○)

◆ 유형 4: 전달인자 없고, 반환 값 없다! 전달인자(X), 반환 값(X)

		반환 값	
		있다	없다
전달인자	있다	유형 1	유형 2
	없다	유형 3	유형 4

01. 함수를 정의하고 선언하기

❖ 전달인자와 반환 값, 둘 다 있는 함수 (유형 1) (1/2)

- 전달인자는 int형 정수 둘이며, 이 둘을 이용한 덧셈을 진행해 보겠습니다.
- 덧셈 결과는 반환이 되며, 따라서 반환형도 int형으로 선언합니다.
- 마지막으로 함수의 이름은 Add라 해보겠습니다.

```
A. B. C.  
int Add (int num1, int num2)  
{  
    int result = num1 + num2;  
D. return result;  
}
```

- A. 반환형
- B. 함수의 이름
- C. 매개변수
- D. 값의 반환

01. 함수를 정의하고 선언하기

❖ 전달인자와 반환 값, 둘 다 있는 함수 (유형 1) (2/2)

```
1  /* func_Add.c */
2  #include <stdio.h>
3
4  int Add(int num1, int num2)
5  {
6      return num1 + num2;
7  }
8
9  int main(void)
10 {
11     int result;
12
13     result = Add(3, 4);
14     printf("덧셈 결과1: %d\n", result);
15
16     result = Add(5, 8);
17     printf("덧셈 결과2: %d\n", result);
18     return 0;
19 }
```


함수 호출이 완료되면 호출한 위치로
이동해서 실행을 이어갑니다.

덧셈 결과1: 7
덧셈 결과2: 13

01. 함수를 정의하고 선언하기

❖ 전달인자는 있지만 반환 값이 없는 경우 (유형 2)

```
1  /* func_ShowAddResult.c */
2  #include <stdio.h>
3
4  void ShowAddResult(int num)
5  {
6      printf("덧셈 결과 출력: %d\n", num);
7  }
8
9  int main(void)
10 {
11     int result;
12
13     result = 3 + 4;
14
15     ShowAddResult(result);
16
17     return 0;
18 }
```




덧셈 결과 출력: 7

01. 함수를 정의하고 선언하기

❖ 전달인자는 없지만 반환 값이 있는 경우 (유형 3)

```
1  /* func_ReadNum.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  int ReadNum(void)
6  {
7      int num;
8      scanf("%d", &num);
9      return num;
10 }
11
12 int main(void)
13 {
14     int result;
15     result = ReadNum();
16     printf("결과 출력: %d\n", result);
17     return 0;
18 }
```



3
결과 출력: 3

01. 함수를 정의하고 선언하기

❖ 전달인자도 없고, 반환 값도 없는 경우 (유형 4)

```
1  /* func_HowToUseThisProg.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  void HowToUseThisProg(void)
6  {
7      printf("두 개의 정수를 입력하시면 덧셈 결과가 출력됩니다.\n");
8      printf("자! 그럼 두 개의 정수를 입력하세요.\n");
9  }
10
11 int main(void)
12 {
13     int num1, num2;
14     HowToUseThisProg();
15     scanf("%d %d", &num1, &num2);
16     printf("덧셈 결과 출력: %d\n", num1 + num2);
17     return 0;
18 }
```

두 개의 정수를 입력하시면 덧셈 결과가 출력됩니다.
자! 그럼 두 개의 정수를 입력하세요.
3 4
덧셈 결과 출력: 7

01. 함수를 정의하고 선언하기

❖ 4가지 함수 유형을 조합한 예제

```
1  /* four_func.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  int Add(int num1, int num2)
6  {
7      return num1 + num2;
8  }
9
10 void ShowAddResult(int num)
11 {
12     printf("덧셈 결과 출력: %d\n", num);
13 }
14
15 int ReadNum(void)
16 {
17     int num;
18     scanf("%d", &num);
19     return num;
20 }
```

```
21 void HowToUseThisProg(void)
22 {
23     printf("두 개의 정수를 입력하시면 덧셈 결과가 출력됩니다.\n");
24     printf("자! 그럼 두 개의 정수를 입력하세요.\n");
25 }
26 int main(void)
27 {
28     int result, num1, num2;
29     HowToUseThisProg();
30     num1 = ReadNum();
31     num2 = ReadNum();
32     result = Add(num1, num2);
33     ShowAddResult(result);
34     return 0;
35 }
```

두 개의 정수를 입력하시면 덧셈 결과가 출력됩니다.
자! 그럼 두 개의 정수를 입력하세요.
3 4
덧셈 결과 출력: 7

01. 함수를 정의하고 선언하기

❖ 값을 반환하지 않는 return

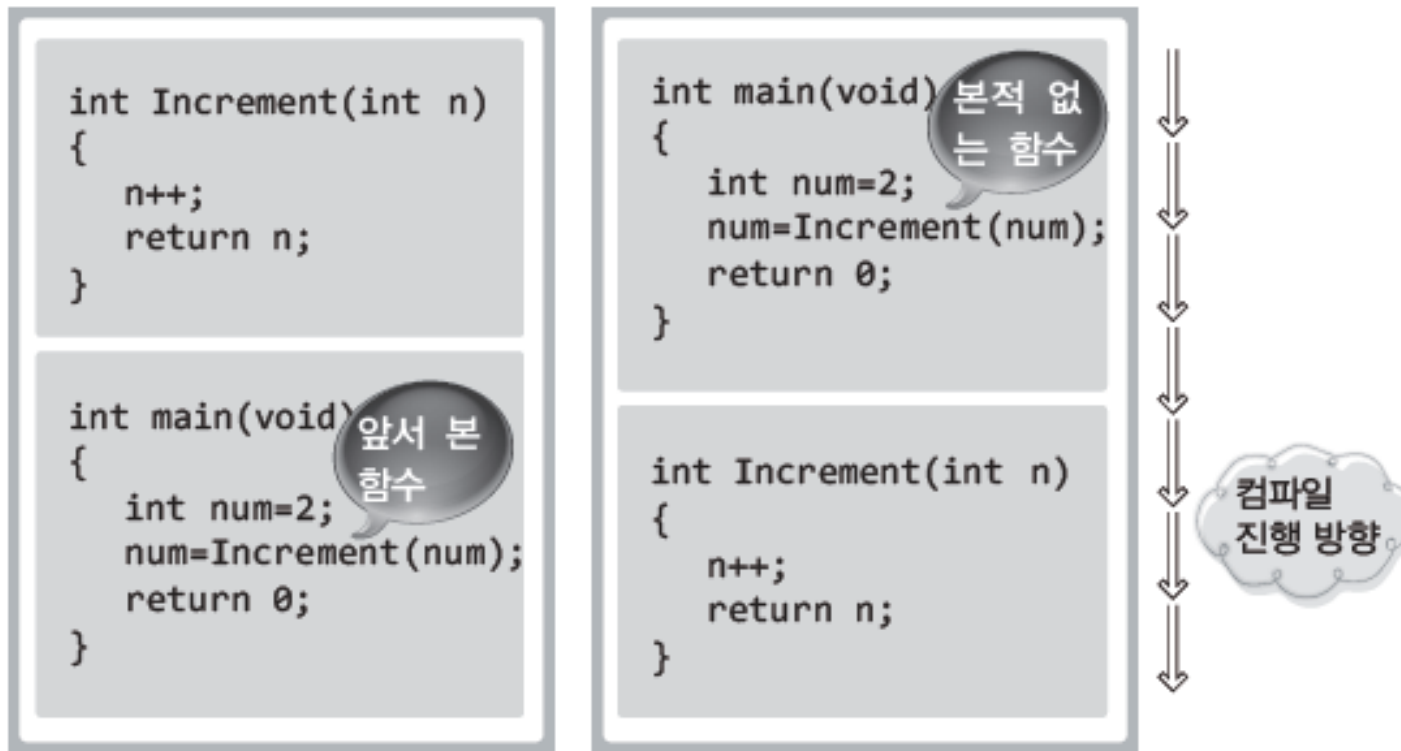
- return문에는 '값의 반환'과 '함수의 탈출'이라는 두 가지 기능이 담겨있습니다.
- 위의 예제에서 보이듯이 값을 반환하지 않는 형태로 return문을 구성하여 값을 반환하지 않되 함수를 빠져나가는 용도로 사용할 수 있습니다.

```
void ShowAddResult(int num)
{
    if (num < 0)
        return;
    ...
}
```

01. 함수를 정의하고 선언하기

❖ 함수의 정의와 그에 따른 원형의 선언 (1/2)

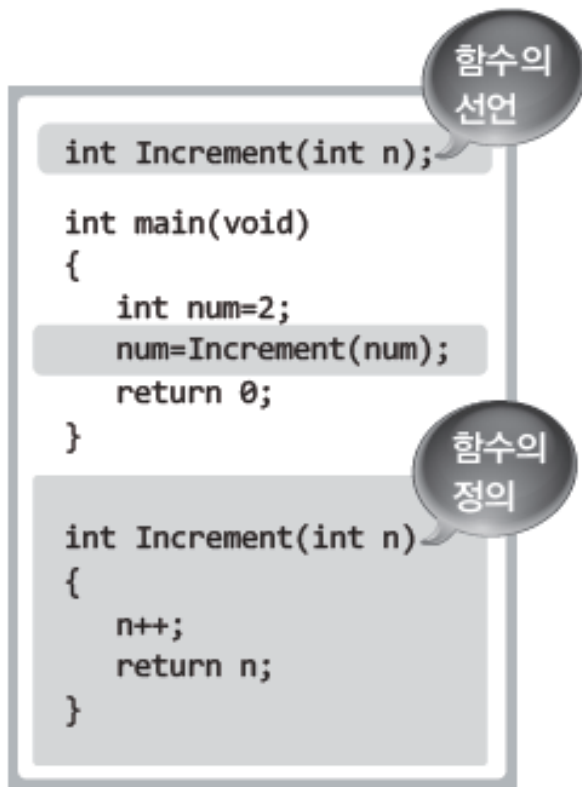
- 컴파일이 위에서 아래로 진행이 되기 때문에 함수의 배치 순서는 중요합니다.
- 컴파일 되지 않은 함수는 호출이 불가능합니다.



01. 함수를 정의하고 선언하기

❖ 함수의 정의와 그에 따른 원형의 선언 (1/2)

- 이후에 등장하는 함수에 대한 정보를 컴파일러에게 제공해서
이후에 등장하는 함수의 호출 문장이 컴파일 가능하게 도울 수 있습니다.
- 이렇게 제공되는 함수의 정보를 가리켜 '함수의 선언'이라 합니다.



```
int Increment(int n);           // 함수의 선언
int Increment(int);             // 위와 동일한 함수 선언, 매개변수 이름 생략 가능
```

01. 함수를 정의하고 선언하기

❖ 다양한 종류의 함수 정의 (1/2)

```
1  /* large_number.c */
2  #include <stdio.h>
3
4  int NumberCompare(int num1, int num2);
5
6  int main(void)
7  {
8      printf("3과 4중에 큰 수는 %d이다.\n", NumberCompare(3, 4));
9      printf("7과 2중에 큰 수는 %d이다.\n", NumberCompare(7, 2));
10     return 0;
11 }
12 int NumberCompare(int num1, int num2)
13 {
14     if (num1 > num2)
15         return num1;
16     else
17         return num2;
18 }
```

3과 4중에 큰 수는 4이다.
7과 2중에 큰 수는 7이다.

01. 함수를 정의하고 선언하기

❖ 다양한 종류의 함수 정의 (2/2)

```
1  /* abso_large.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  int AbsoCompare(int num1, int num2);
6  int GetAbsoValue(int num);
7
8  int main(void)
9  {
10     int num1, num2;
11     printf("두 개의 정수 입력: ");
12     scanf("%d %d", &num1, &num2);
13     printf("절댓값이 큰 정수: %d\n", AbsoCompare(num1, num2));
14     return 0;
15 }
16
```

```
17 int AbsoCompare(int num1, int num2)
18 {
19     if (GetAbsoVal(num1) > GetAbsoVal(num2))
20         return num1;
21     else
22         return num2;
23 }
24
25 int GetAbsoValue(int num)
26 {
27     if (num < 0)
28         return num * (-1);
29     else
30         return num;
31 }
```

두 개의 정수 입력: 5 -9
절댓값이 큰 정수: -9

02. 변수의 존재 기간과 접근 범위: 지역변수

- 01. 함수를 정의하고 선언하기
- 03. 전역변수, static 변수, register 변수
- 04. 재귀함수에 대한 이해
- 05. 연습 문제

02. 변수의 존재 기간과 접근 범위: 지역변수

❖ 함수 내에만 존재 및 접근 가능한 지역변수

```
1  #include <stdio.h>
2  int FuncOne(void)
3  {
4      int num = 10;    // 이후부터 FuncOne의 num 유효
5      num++;
6      printf("FuncOne num: %d\n", num);
7      return 0;        // FuncOne의 num이 유효한 마지막 문장
8  }
9  int FuncTwo(void)
10 {
11     int num1 = 20;    // 이후부터 num1 유효
12     int num2 = 30;    // 이후부터 num2 유효
13     num1++, num2--;
14     printf("num1 & num2: %d %d\n", num1, num2);
15     return 0;        // num1, num2이 유효한 마지막 문장
16 }
```

```
17 int main(void)
18 {
19     int num = 17;    // 이후부터 main의 num 유효
20     FuncOne();
21     FuncTwo();
22     printf("main num: %d\n", num);
23     return 0;        // main의 num이 유효한 마지막 문장
24 }
```

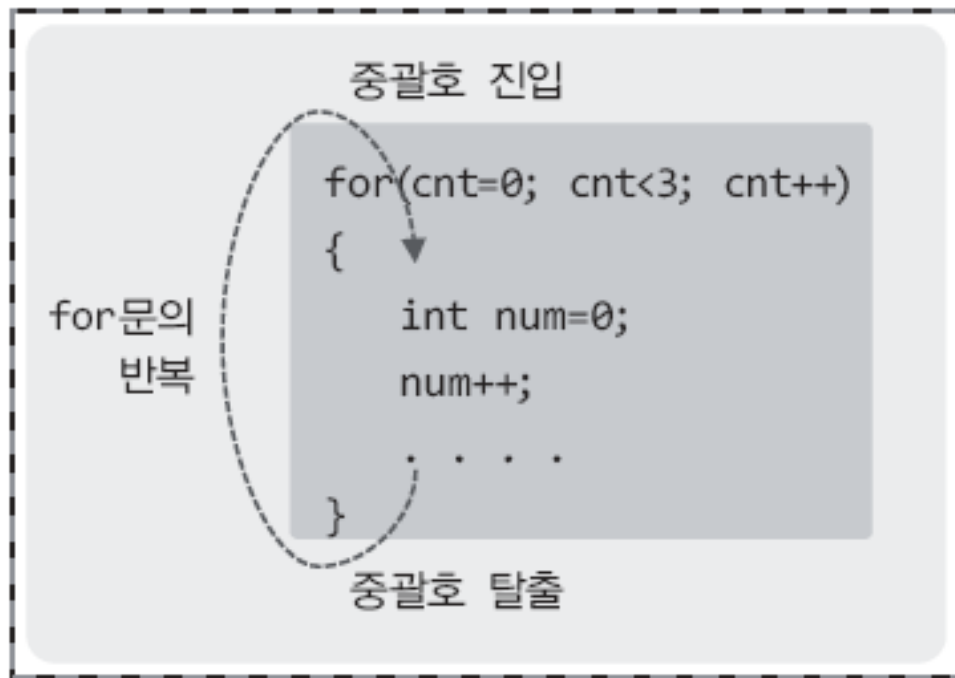
FuncOne num: 11
num1 & num2: 21 29
main num: 17

- ✓ 함수 내에 선언되는 변수를 가리켜 지역변수라고 합니다.
- ✓ 지역변수는 선언된 이후로부터 함수 내에서만 접근이 가능합니다.
- ✓ 한 지역(함수) 내에 동일한 이름의 변수 선언 불가능합니다.
- ✓ 다른 지역에 동일한 이름의 변수 선언 가능합니다.
- ✓ 해당 지역을 빠져나가면 지역변수는 소멸됩니다.
- ✓ 그리고 호출될 때마다 새롭게 할당됩니다.

02. 변수의 존재 기간과 접근 범위: 지역변수

❖ 다양한 형태의 지역변수 (1/3)

- for문의 중괄호 내에 선언된 변수도 지역변수입니다.
- 그리고 이 지역변수는 for문의 중괄호를 빠져나가면 소멸됩니다.
- 따라서 for문의 반복 횟수만큼 지역변수가 할당되고 소멸됩니다.



02. 변수의 존재 기간과 접근 범위: 지역변수

❖ 다양한 형태의 지역변수 (2/3)

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int num = 1;
6      if (num == 1)
7      {
8          int num = 7;  // 이 행을 주석 처리하였을 때, 실행 결과를 확인해 보세요.
9          num += 10;
10         printf("if문 내 지역변수 num: %d\n", num);
11     }
12     printf("main 함수 내 지역변수 num: %d\n", num);
13     return 0;
14 }
```

if문 내 지역변수 num: 17
main 함수 내 지역변수 num: 1

if문 내 지역변수 num: 11
main 함수 내 지역변수 num: 11

주석처리 후 실행 결과

02. 변수의 존재 기간과 접근 범위: 지역변수

❖ 다양한 형태의 지역변수 (3/3)

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int num = 1;
6
7      {    // 중괄호 시작
8          int num = 7; // 이 행을 주석 처리하였을 때, 실행 결과를 확인해 보세요.
9          num += 10;
10         printf("중괄호 내 지역변수 num: %d\n", num);
11     }    // 중괄호 끝
12     printf("main 함수 내 지역변수 num: %d\n", num);
13     return 0;
14 }
```

아무 의미 없는 중괄호를 통해서도 지역변수가 만들어진다는 것을 확인하기 위한 예제입니다.

중괄호 내 지역변수 num: 17
main 함수 내 지역변수 num: 1

중괄호 내 지역변수 num: 11
main 함수 내 지역변수 num: 11

주석처리 후 실행 결과

03. 전역변수, static 변수, register 변수


- 01. 함수를 정의하고 선언하기
- 02. 변수의 존재 기간과 접근 범위: 지역변수
- 04. 재귀함수에 대한 이해
- 05. 연습 문제

03. 전역변수, static 변수, register 변수

❖ 전역변수의 이해와 선언 방법 (1/2)

```
1  #include <stdio.h>
2
3  void Add(int val);
4  int num;    // 전역변수는 기본 0으로 초기화됩니다.
5
6  int main(void)
7  {
8      printf("num: %d\n", num);
9      Add(3);
10     printf("num: %d\n", num);
11     num++;    // 전역변수 num의 값 1증가
12     printf("num: %d\n", num);
13     return 0;
14 }
15
16 void Add(int val)
17 {
18     num += val;    // 전역변수 num의 값 val만큼 증가
19 }
```

- ✓ 전역변수는 함수 외부에 선언됩니다.
- ✓ 프로그램의 시작과 동시에 메모리 공간에 할당되어 종료 시까지 존재합니다.
- ✓ 별도의 값으로 초기화하지 않으면 0으로 초기화됩니다.
- ✓ 프로그램 전체 영역 어디서든 접근이 가능합니다.




num: 0
num: 3
num: 4

03. 전역변수, static 변수, register 변수

❖ 전역변수의 이해와 선언 방법 (2/2)

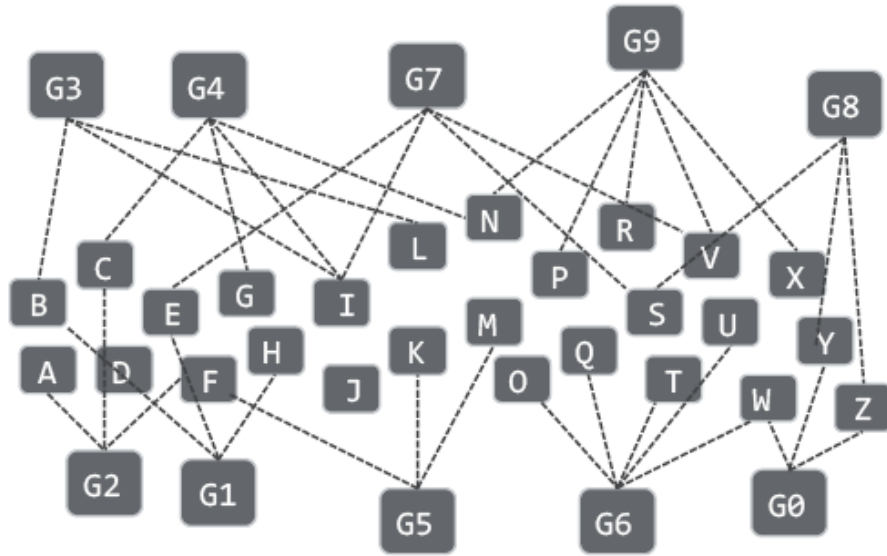
```
1  #include <stdio.h>
2
3  int Add(int val);
4  int num = 1;
5
6  int main(void)
7  {
8      int num = 5;
9      printf("num: %d\n", Add(3));
10     printf("num: %d\n", num + 9);
11     return 0;
12 }
13
14 int Add(int val)
15 {
16     int num = 9;
17     num += val;
18     return num;
19 }
```



```
num: 12
num: 14
```

03. 전역변수, static 변수, register 변수

❖ 전역변수! 많이 써도 되나요?



G0~G9의 전역변수와 함수와의 접근 관계의 예시

- ✓ 전역변수를 많이 쓰면 좋지 않습니다.
- ✓ 전역변수의 변경은 전체 프로그램의 변경으로 이어질 수 있으며 전역변수에 의존적인 코드는 프로그램 전체 영역에서 찾아야 합니다. (어디서든 접근이 가능한 변수이므로)

03. 전역변수, static 변수, register 변수

❖ 지역변수에 static 선언을 추가한 static 변수

```
1  #include <stdio.h>
2  void SimpleFun(void)
3  {
4      static int num1 = 0;  // 초기화하지 않으면 0으로 초기화
5      int num2 = 0;        // 초기화하지 않으면 쓰레기 값 초기화
6      num1++, num2++;
7      printf("static: %d, local: %d\n", num1, num2);
8  }
9  int main(void)
10 {
11     int j;
12     for (j = 0; j < 3; j++)
13         SimpleFunc();
14     return 0;
15 }
```

- static
(형용사) 고정된, 고정적인
(형용사) 정지 상태의

static 변수의 특성

- ✓ (지역변수 특성):
선언된 함수 내에서만 접근이 가능합니다.
- ✓ (전역변수 특성):
딱 1회 초기화되고 프로그램 종료 시까지 메모리 공간에 존재합니다.

아무 곳에서 접근할 수 있는 전역변수와 달리,
접근의 범위를 SimpleFunc로 제한하기 위해서 사용됩니다.

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

03. 전역변수, static 변수, register 변수

❖ static 변수는 좀 써도 되나요?

- 전역변수가 필요한 이유 중 하나는 다음과 같습니다.
 - ✓ 선언된 변수가 함수를 빠져나가도 계속해서 메모리 공간에 존재할 필요가 있을 경우
- 함수를 빠져나가도 계속해서 메모리 공간에 존재해야 하는 변수를 선언하는 방법은 다음 두 가지입니다.
 - ✓ 전역변수, static 변수
- static 변수는 접근의 범위가 전역변수보다 훨씬 좁기 때문에 훨씬 안정적입니다.
 - ✓ static 변수를 사용하여 전역변수의 선언을 최소화하는 것이 좋습니다.

03. 전역변수, static 변수, register 변수

❖ 보다 빠르게! register 변수

```
int SoSimple(void)
{
    int num1 = 2;
    register int num2 = 3;
    ...
}
```

- ✓ 변수를 선언함에 있어서 register 키워드를 붙여주면, 변수 num2는 CPU의 레지스터라는 메모리 영역에 저장됩니다.
- ✓ 레지스터는 CPU의 접근이 가장 빠른 메모리 공간입니다.
- ✓ 따라서 변수 num1보다 num2가 빠르게 처리될 것입니다.
- ✓ CPU의 레지스터는 그 크기가 제한되어 있는 메모리 공간입니다.
- ✓ 따라서 이 공간에 변수를 선언하는 것이 여의치 않을 경우, 컴파일러는 이 선언을 무시하기도 합니다.

- ✓ 생성과 소멸이 빈번한 변수를 register 변수로 선언하게 되면 많은 성능 향상을 기대할 수 있습니다.
- ✓ 하지만 우리는 register 변수 선언과 관련해서 고민할 필요는 없습니다.
- ✓ 대부분의 컴파일러가 코드 최적화라는 것을 수행하면서, 알아서 register 키워드를 붙여줍니다.

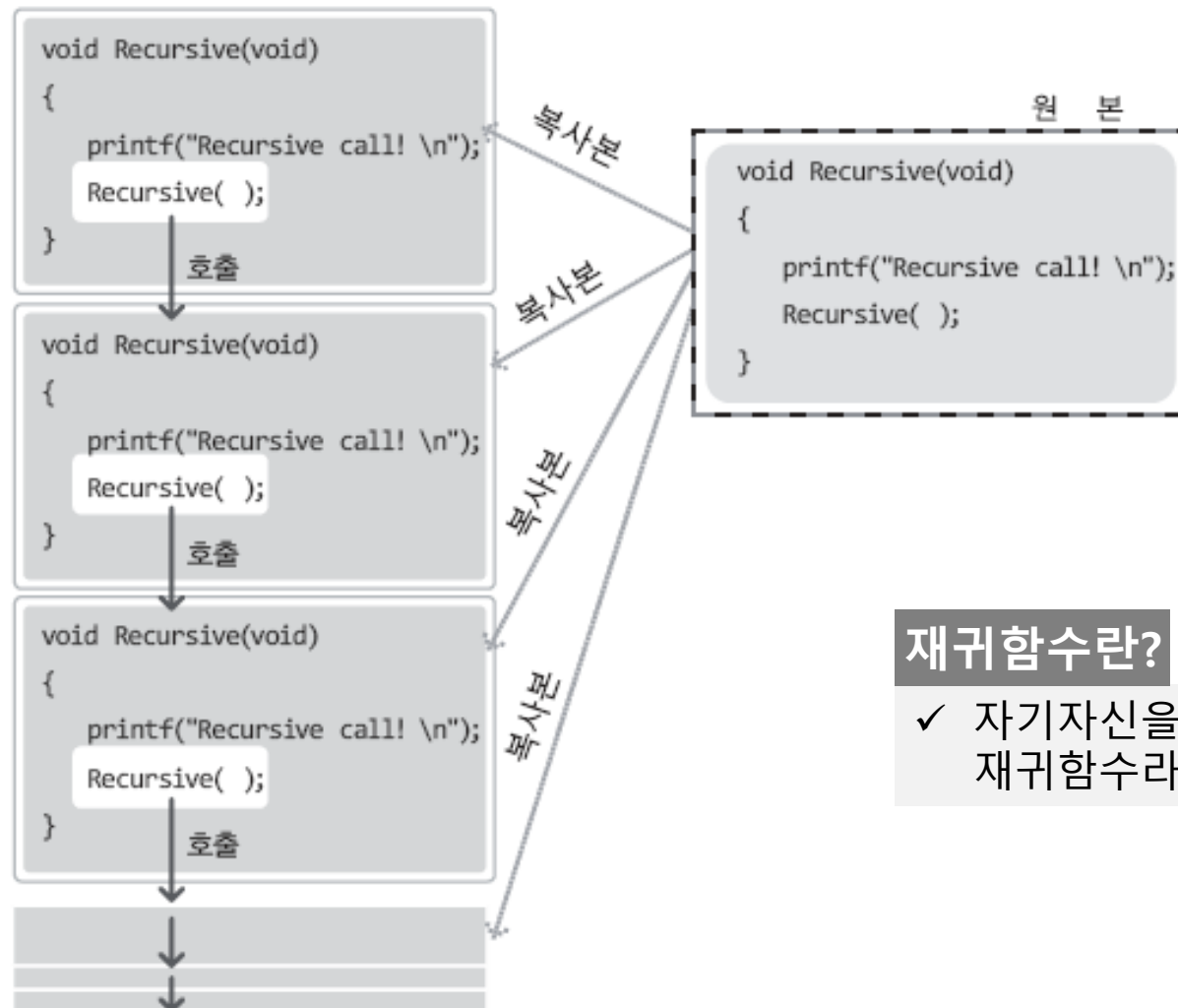


04. 재귀함수에 대한 이해

- 01. 함수를 정의하고 선언하기
- 02. 변수의 존재 기간과 접근 범위: 지역변수
- 03. 전역변수, static 변수, register 변수
- 05. 연습 문제

04. 재귀함수에 대한 이해

❖ 재귀함수의 기본적인 이해



재귀함수란?

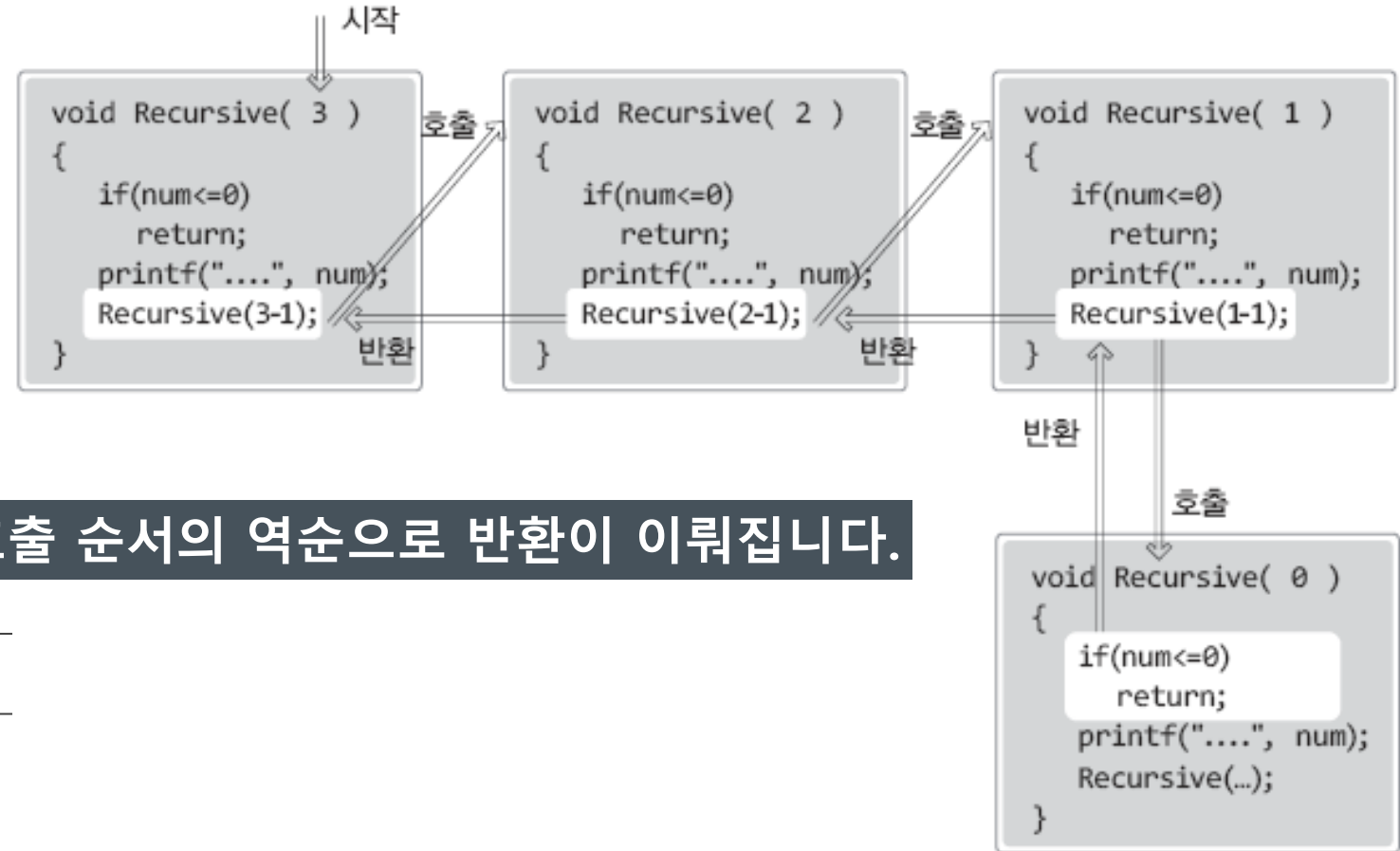
- ✓ 자기자신을 재호출하는 형태로 정의된 함수를 가리켜 재귀함수라고 합니다.

04. 재귀함수에 대한 이해

❖ 탈출조건이 존재하는 재귀함수의 예

```
1  #include <stdio.h>
2  void Recursive(int num)
3  {
4      if (num <= 0)    // 재귀의 탈출 조건
5          return;    // 재귀의 탈출!
6      printf("Recursive call! %d\n", num);
7      Recursive(num - 1);
8  }
9  int main(void)
10 {
11     Recursive(3);
12     return 0;
13 }
```

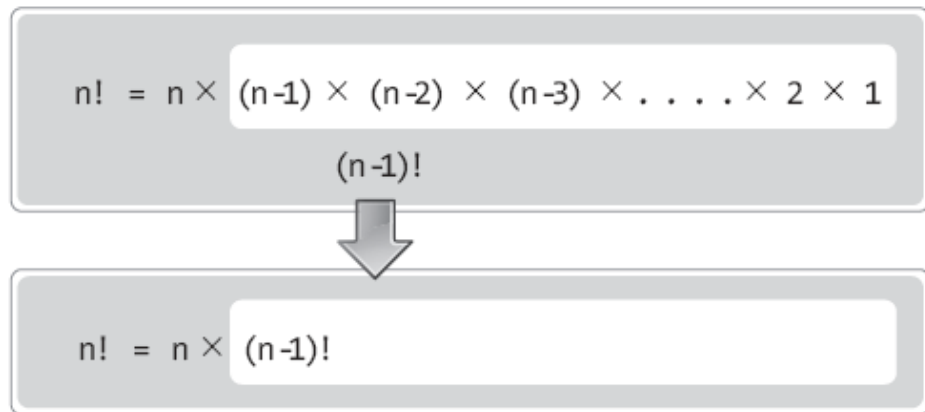
```
Recursive call! 3
Recursive call! 2
Recursive call! 1
```



호출 순서의 역순으로 반환이 이뤄집니다.

04. 재귀함수에 대한 이해

❖ 재귀함수의 활용 사례



팩토리얼에 대한 수학적 표현

$$f(n) = \begin{cases} n \times f(n-1), & \text{if } 1 \leq n \\ 1, & \text{if } n = 0 \end{cases}$$

$n \times f(n-1)$ 에 대한 코드 구현

```
if (n >= 1)
    return n * Factorial(n - 1);
```

$f(n) = 1$ 에 대한 코드 구현


```
if (n == 0)
    return 1;
```

```
if (n >= 1)
    return n * Factorial(n - 1);
else
    return 1;
```

04. 재귀함수에 대한 이해

❖ 팩토리얼 함수의 예

```
1  #include <stdio.h>
2  int Factorial(int n)
3  {
4      if (num == 0)
5          return 1;
6      else
7          return n * Factorial(n - 1);
8  }
9  int main(void)
10 {
11     printf("1! = %d\n", Factorial(1));
12     printf("2! = %d\n", Factorial(2));
13     printf("3! = %d\n", Factorial(3));
14     printf("4! = %d\n", Factorial(4));
15     printf("5! = %d\n", Factorial(5));
16     return 0;
17 }
```



```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
```

**C언어의 재귀함수를 이용하면 재귀적으로
작성된 식을 그대로 코드로 옮길 수 있습니다.**

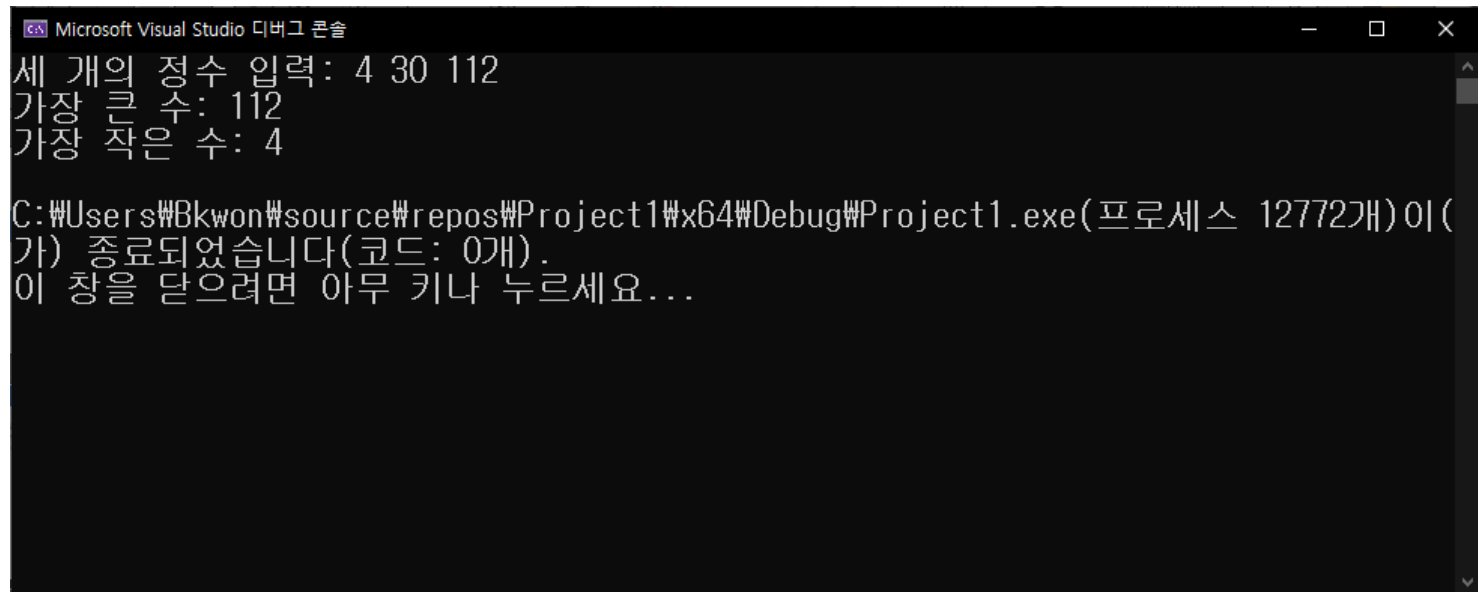
05. 연습 문제

- 01. 함수를 정의하고 선언하기
- 02. 변수의 존재 기간과 접근 범위: 지역변수
- 03. 전역변수, static 변수, register 변수
- 04. 재귀함수에 대한 이해

05. 연습 문제

❖ 연습 문제 1.

- 세 개의 정수를 입력받아서 그 중 가장 큰 수를 반환하는 함수와 가장 작은 수를 반환하는 함수를 만들어 보세요.
그리고 이 함수들을 이용하는 적절한 main 함수도 구현해 보세요.



```
Microsoft Visual Studio 디버그 콘솔
세 개의 정수 입력: 4 30 112
가장 큰 수: 112
가장 작은 수: 4

C:\Users\Bkwon\source\repos\Project1\x64\Debug\Project1.exe(프로세스 12772개)이(
가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

05. 연습 문제

❖ 연습 문제 1. 정답 및 해설

```
1  /* example1.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  int Max(int a, int b, int c);
6  int Min(int a, int b, int c);
7
8  int main(void)
9  {
10     int num1, num2, num3;
11     printf("세 개의 정수 입력: ");
12     scanf("%d %d %d", &num1, &num2, &num3);
13
14     printf("가장 큰 수: %d\n", Max(num1, num2, num3));
15     printf("가장 작은 수: %d\n", Min(num1, num2, num3));
16     return 0;
17 }
```

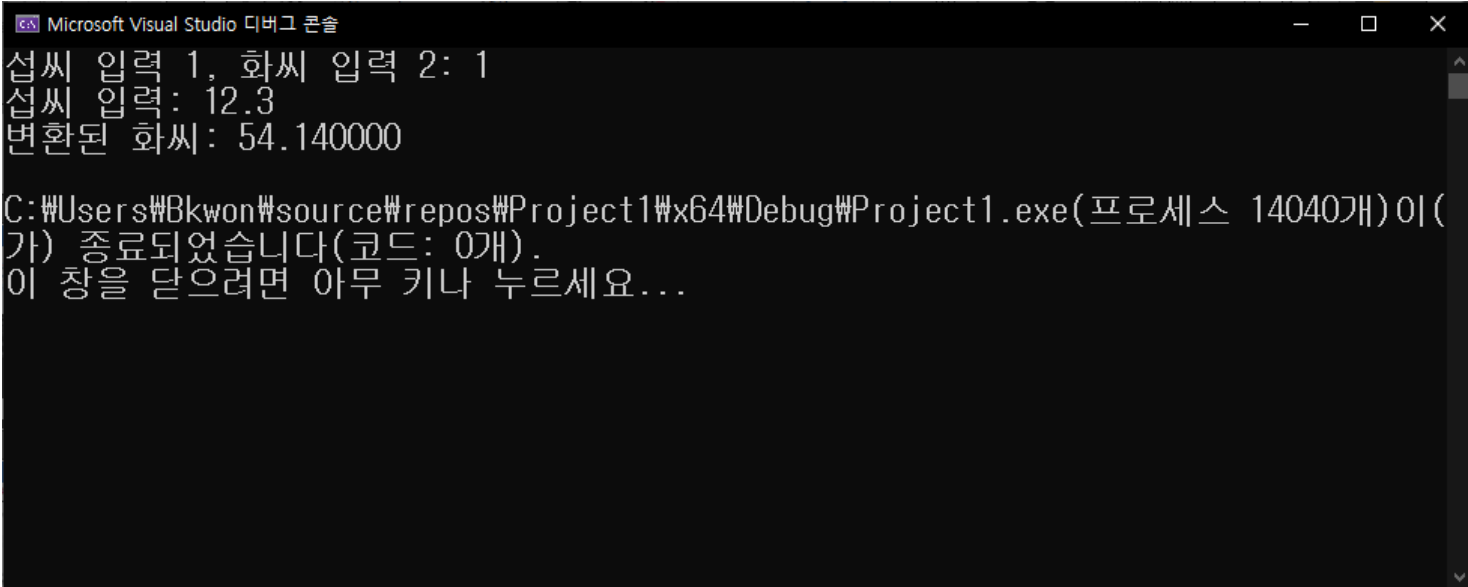
```
18 int Max(int a, int b, int c)
19 {
20     if (a > b)
21         return (a > c) ? a:c;
22     else
23         return (b > c) ? b:c;
24 }
25
26 int Min(int a, int b, int c)
27 {
28     if (a < b)
29         return (a < c) ? a:c;
30     else
31         return (b < c) ? b:c;
32 }
```

05. 연습 문제

❖ 연습 문제 2.

- 섭씨 온도를 입력하면 화씨 온도로 변환하는 Fahrenheit라는 이름의 함수와 그 반대로 화씨 온도를 섭씨 온도로 변환하는 Celsius 함수를 정의하고 이에 적절한 main 함수를 구현해 보세요.
섭씨와 화씨 온도 사이의 변환 공식은 다음과 같습니다.

$$F = 1.8 \times C + 32$$



```
Microsoft Visual Studio 디버그 콘솔
섭씨 입력 1, 화씨 입력 2: 1
섭씨 입력: 12.3
변환된 화씨: 54.140000

C:\Users\Bkwon\source\repos\Project1\x64\Debug\Project1.exe(프로세스 14040개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```


05. 연습 문제

❖ 연습 문제 2. 정답 및 해설

```
1  /* example2.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  double Fahrenheit(double c)    // 섭씨 to 화씨
6  {
7      return 1.8 * c + 32;
8  }
9  double Celsius(double f)      // 화씨 to 섭씨
10 {
11     return (f - 32) / 1.8;
12 }
13 int main(void)
14 {
15     int sel;
16     double val;
17     printf("섭씨 입력 1, 화씨 입력 2: ");
18     scanf("%d", &sel);
```

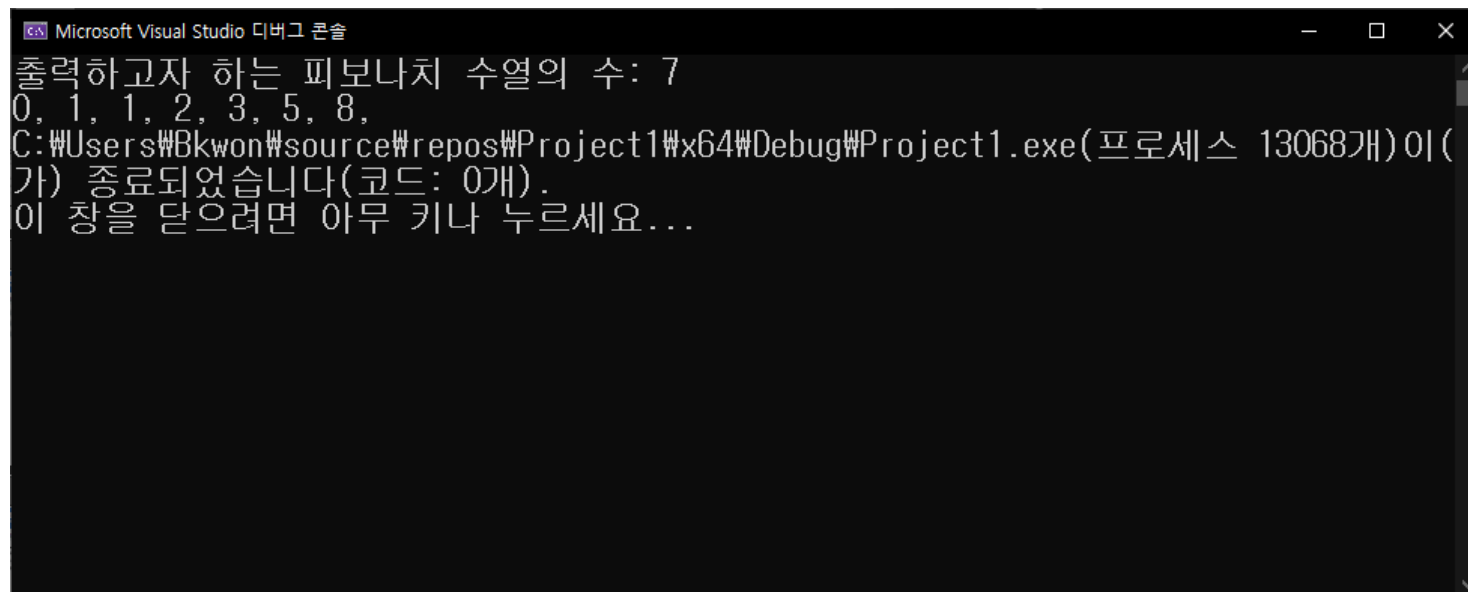
```
19     if (sel == 1)
20     {
21         printf("섭씨 입력: ");
22         scanf("%lf", &val);
23         printf("변환된 화씨: %f\n", Fahrenheit(val));
24     }
25     else if (sel == 2){
26         printf("화씨 입력: ");
27         scanf("%lf", &val);
28         printf("변환된 섭씨: %f\n", Celsius(val));
29     }
30     else
31         printf("선택 오류\n");
32     return 0;
33 }
```

05. 연습 문제

❖ 연습 문제 3.

- 피보나치 수열을 출력하는 함수를 구현해 보세요. 예를 들어 사용자로부터 5라는 숫자를 입력받으면 0부터 시작해서 총 다섯 개의 피보나치 수열을 출력해야 합니다. 피보나치 수열은 다음과 같은 수열을 의미하는 것입니다.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...



```
Microsoft Visual Studio 디버그 콘솔
출력하고자 하는 피보나치 수열의 수: 7
0, 1, 1, 2, 3, 5, 8,
C:\Users\Bkwon\source\repos\Project1\x64\Debug\Project1.exe(프로세스 13068개)이(
가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

05. 연습 문제

❖ 연습 문제 3. 정답 및 해설

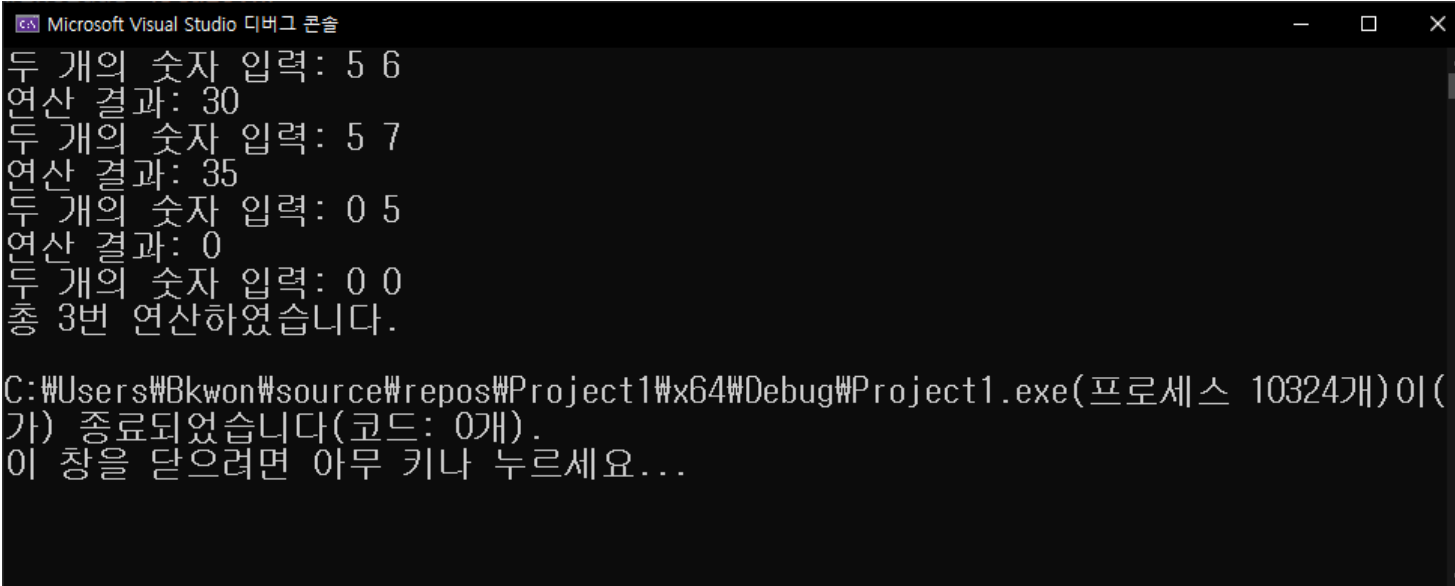
```
1  /* example3.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  void ShowFibo(int a);
6
7  int main(void)
8  {
9      int n;
10     printf("출력하고자 하는 피보나치 수열의 수: ");
11     scanf("%d", &n);
12     if (n < 1)
13     {
14         printf("1 이상의 값을 입력하세요.\n");
15         return -1;
16     }
17     ShowFibo(n);
18     return 0;
19 }
```

```
20 void ShowFibo(int a)
21 {
22     int f1 = 0;
23     int f2 = 1;
24     int f3, j;
25
26     if (a == 1)
27         printf("%d, ", f1);
28     else
29         printf("%d, %d, ", f1, f2);
30
31     for (j = 0; j < a - 2; j++)
32     {
33         f3 = f1 + f2;
34         printf("%d, ", f3);
35         f1 = f2;
36         f2 = f3;
37     }
38 }
```

05. 연습 문제

❖ 연습 문제 4.

- 곱셈 기능을 지니는 함수를 하나 구현하고 main 함수에서 이를 호출하는 형태로 프로그램을 구성해 보세요. main 함수에서는 사용자로부터 두 개의 숫자를 입력받아서 곱셈 결과를 출력해 줘야 합니다. 이러한 작업은 사용자가 0을 두 개 입력할 때까지 계속되어야 합니다. 그리고 프로그램이 종료되면 연산을 몇 번 하였는지도 출력해 줘야 합니다.



```
Microsoft Visual Studio 디버그 콘솔
두 개의 숫자 입력: 5 6
연산 결과: 30
두 개의 숫자 입력: 5 7
연산 결과: 35
두 개의 숫자 입력: 0 5
연산 결과: 0
두 개의 숫자 입력: 0 0
총 3번 연산하였습니다.

C:\Users\Bkwon\source\repos\Project1\x64\Debug\Project1.exe(프로세스 10324개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

05. 연습 문제

❖ 연습 문제 4. 정답 및 해설

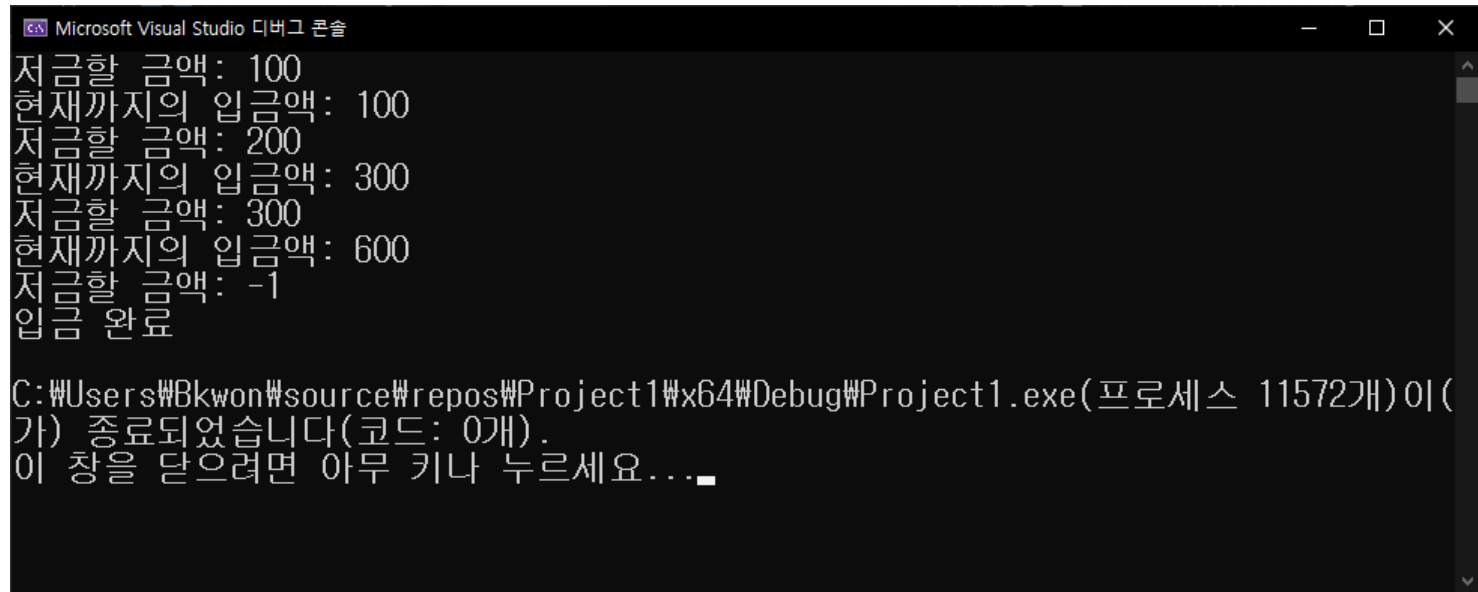
```
1  /* example4.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  int count = 0;
6  int Multiplication(int num1, int num2);
7
8  int main(void)
9  {
10     while (1)
11     {
12         int num1, num2;
13         printf("두 개의 숫자 입력: ");
14         scanf("%d %d", &num1, &num2);
15         if ((num1 == 0) && (num2 == 0))
16             break;
17
18         printf("연산 결과: %d\n", Multiplication(num1, num2));
19     }
```

```
20     printf("총 %d번 연산하였습니다.\n", count);
21     return 0;
22 }
23
24 int Multiplication(int num1, int num2)
25 {
26     count++;
27     return num1 * num2;
28 }
```

05. 연습 문제

❖ 연습 문제 5.

- 저금통 기능을 지니는 함수를 구현해 보세요. 이 함수는 호출 시 전달되는 인자 값을 저금통처럼 누적 시킵니다. 그리고 누적된 금액을 출력해 줍니다. -1이 입력될 때까지 계속해서 진행하도록 main 함수를 구성하세요. 단 한 가지 제한 사항이 있습니다. 이 프로그램에서는 전역 변수를 사용하지 않기로 하겠습니다. 필요하다면 static 지역 변수를 사용하세요.



```
Microsoft Visual Studio 디버그 콘솔
저금할 금액: 100
현재까지의 입금액: 100
저금할 금액: 200
현재까지의 입금액: 300
저금할 금액: 300
현재까지의 입금액: 600
저금할 금액: -1
입금 완료

C:\Users\Bkwon\source\repos\Project1\x64\Debug\Project1.exe(프로세스 11572개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

05. 연습 문제

❖ 연습 문제 5. 정답 및 해설

```
1  /* example5.c */
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <stdio.h>
4
5  void save(int money);
6  int main(void)
7  {
8      while (1)
9      {
10         int money;
11         printf("저금할 금액: ");
12         scanf("%d", &money);
13         if (money == -1)
14             break;
15         save(money);
16     }
17     printf("입금 완료\n");
18     return 0;
19 }
```

```
20 void save(int money)
21 {
22     static int total;
23     total = total + money;
24     printf("현재까지의 입금액: %d\n", total);
25 }
```

- ❖ 01. 함수를 정의하고 선언하기
- ❖ 02. 변수의 존재 기간과 접근 범위: 지역변수
- ❖ 03. 전역변수, static 변수, register 변수
- ❖ 04. 재귀함수에 대한 이해
- ❖ 05. 연습 문제

THANK YOU!

Q & A

- Name: 권범
- Office: 동덕여자대학교 인문관 B821호
- Phone: 02-940-4752
- E-mail: bkwon@dongduk.ac.kr