한빛아카데미 Hanbit Academy, Inc.

C 언어 프로그래밍

Part 02. C 언어 기본 학습

Chapter 08. 함수

목차

학습목표

- **함수를 정의하는 형식과 구성 요소를 살펴본다.**
- **실인자와 매개변수의 개념을 이해하고 차이점을 살펴본다.**
- **값 전달 방식과 참조 전달 방식을 이해한다.**
- **재귀 호출의 장점과 주의할 점을 살펴본다.**
- **객체의 가시 범위와 생명 주기의 개념을 이해한다.**

01
함수의 정의와 선언

01. 함수의 정의와 선언

I. 함수의 정의

- **함수의 정의 형식**

```
반환형 함수명(매개변수타입1 매개변수1, ···, 매개변수타입N 매개변수N)
{
        명령절(0개 이상의 명령문 모임)
}
```

- **함수 정의 형식의 구성 요소**
  - 반환형 : 함수가 반환될 때 반환 값의 타입
  - 함수명 : 함수의 이름으로 식별자 규칙을 따름
  - 매개변수타입 : 함수를 호출하면서 데이터를 함수에 전달할 때 전달되는 데이터를 인자라고 하는데, 인자를 전달받는 매개변수의 타입
  - 매개변수 : 매개변수의 이름

01. 함수의 정의와 선언

I. 함수의 정의

- 함수 정의하기

[코드 8-1] 급수 함수

```
01  #include <stdio.h>
02
03  int Sum(int from, int to)
04  {
05      int sum = 0;
06      for(int i = from; i <= to; i++)
07          sum += i;
08
09      return sum;
10      printf("Complete!");
11  }
12
13  int main( )
14  {
15      int result = Sum(1, 100);
16      printf("Sum: %d", result);
17  }
```

```
Sum: 5050
```

01. 함수의 정의와 선언

I. 함수의 정의

- **void 함수**
  - 함수가 반환할 때 반환 값이 없는 경우 반환 타입으로 void를 사용함

[코드 8-2] void 함수

```
01  #include <stdio.h>
02
03  void f( )
04  {
05     printf("f 함수");
06  }
07
08  void g(void)
09  {
10     return;
11     printf("g 함수");
12  }
```

01. 함수의 정의와 선언

I. 함수의 정의

- **void 함수**
  - 함수가 반환할 때 반환 값이 없는 경우 반환 타입으로 void를 사용함

[코드 8-2] void 함수

```
13
14  int main( )
15  {
16      f();
17      g();
18  }
```

```
f 함수
```

01. 함수의 정의와 선언

I. 함수의 정의

- **함수 중복 정의**
  - 함수를 이름으로 구분하기 때문에 같은 이름으로 중복 정의하는 것을 허용하지 않음

[코드 8-3] 함수 중복 정의

```
01  void func( )
02  {
03  }
04
05  void func(int arg)   <- 오류
06  {
07  }
08
09  int func()   <- 오류
10  {
11     return 0;
12  }
```

01. 함수의 정의와 선언

II. 함수의 선언

- **함수 선언의 개념**
  - 함수 정의에서 함수 본체를 없애고 끝에 세미콜론(;)이 붙는 선언문

```
반환형 함수명(매개변수타입1 매개변수1, ···, 매개변수타입N 매개변수N);
```

그림 8-1 미리 휴게소를 알려주는 표지판

01. 함수의 정의와 선언

III. 함수를 선언하지 않을 경우

- 함수 선언이 필요한 이유

[코드 8-4] 경고와 오류

```
01  int main( )
02  {
03      func(); ← 경고와 오류 발생
04  }
```

01. 함수의 정의와 선언

III. 함수를 선언하지 않을 경우

- 함수 선언이 필요한 이유

[코드 8-5] 경고

```
01  int main( )
02  {
03     func();      // 경고
04  }
05
06  int func( )
07  {
08     return 1;
09  }
```

01. 함수의 정의와 선언

III. 함수를 선언하지 않을 경우

- 함수 선언이 필요한 이유

[코드 8-6] 재정의 오류

```
01  int main( )
02  {
03     func();      // 경고
04  }
05
06  void func()  <-- 오류
07  {
08  }
```

01. 함수의 정의와 선언

III. 함수를 선언하지 않을 경우

- 해결방법 ❶ 함수를 먼저 정의하기

[코드 8-7] 함수를 먼저 정의하기

```
01  void func()
02  {
03  }
04
05  int main( )
06  {
07     func();
08  }
```

01. 함수의 정의와 선언

III. 함수를 선언하지 않을 경우

- 해결방법 ❷ 함수 선언하기

[코드 8-8] 함수 정의 & 함수 호출

```
01  void func();
02
03  int main( )
04  {
05      func();
06  }
07
08  void func()
09  {
10  }
```

함수 선언

함수 호출

함수 정의

01. 함수의 정의와 선언

IV. 함수 선언이 반드시 필요한 경우

- 함수가 서로를 호출하는 경우

[코드 8-9] 서로 호출하는 함수

```
01  void funcA( )
02  {
03     funcB();     // 경고
04  }
05
06  void funcB()
07  {
08     funcA( );
09  }
10
11  int main( )
12  {
13     funcA();
14  }
```

오류

01. 함수의 정의와 선언

IV. 함수 선언이 반드시 필요한 경우

- 함수가 서로를 호출하는 경우

[코드 8-10] 서로 호출하는 함수 오류 해결

```
01  void funcB();
02
03  void funcA( )
04  {
05     funcB();
06  }
07
08  void funcB()
09  {
10     funcA( );
11  }
12
13  int main( )
14  {
15     funcA();
16  }
```

01. 함수의 정의와 선언

IV. 함수 선언이 반드시 필요한 경우

- 소스 파일이 여러 곳에 흩어져 정의되는 경우

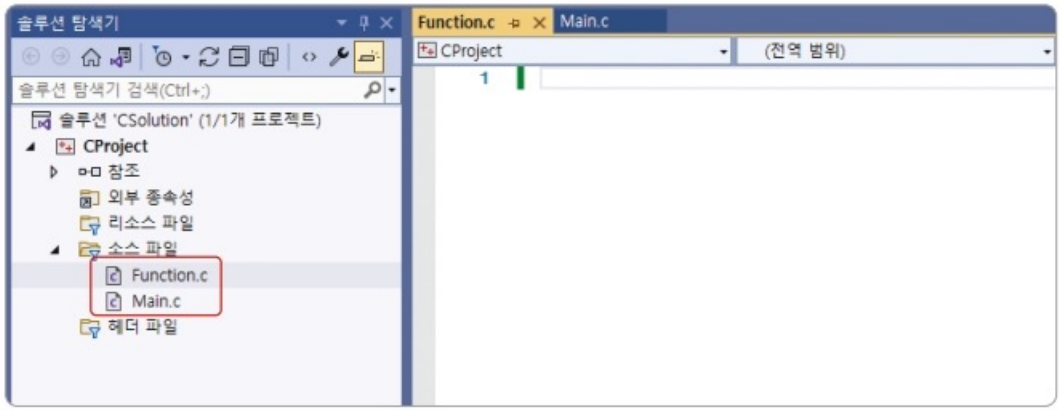


그림 8-2 여러 개의 소스 파일을 사용하는 경우

01. 함수의 정의와 선언

IV. 함수 선언이 반드시 필요한 경우

- 소스 파일이 여러 곳에 흩어져 정의되는 경우

[코드 8-11] 여러 개의 소스 파일

Function.c

```
01  int* func()
02  {
03      return 0;
04  }
```

Main.c

```
01  int main()
02  {
03      *func();      // 오류 발생
04  }
```

01. 함수의 정의와 선언

IV. 함수 선언이 반드시 필요한 경우

- 소스 파일이 여러 곳에 흩어져 정의되는 경우

[코드 8-12] 함수 선언된 Main.c Main.c

```
01 // Main.c
02 int* func();
03
04 int main()
05 {
06    *func();    // 문제 없음
07 }
```

01. 함수의 정의와 선언

IV. 함수 선언이 반드시 필요한 경우

하나 더 알기 **헤더 파일의 필요성**

- 함수를 호출하는 파일 가장 위쪽에는 해당 함수의 선언을 하는 것이 일반적임
- 그러나 함수의 종류가 많고 파일 위쪽에 선언되는 함수가 많아지면 가독성이 떨어져 헤더 파일을 사용하기도 함
- 헤더 파일 덕분에 간결한 소스 코드를 유지할 수 있음

01. 함수의 정의와 선언

확인문제1

1. 다음 빈칸에 들어갈 단어를 채우시오.

함수 정의 형식에서 함수 [ ]은 식별자로서 함수를 구분하는 용도로 사용된다. 특히 C 언어는 함수의 [ ]를 허용하지 않는다. 함수의 본체는 블록으로 감싸진 [ ]이며, 함수에서 명시적으로 반환을 할 경우 [ ]을 사용한다. 만일 함수가 반환 값을 가지지 않을 경우 함수의 반환 타입은 [ ]이다.

2. 다음 빈칸에 들어갈 단어를 채우시오.

함수의 선언은 함수의 정의에서 함수 본체인 [ ]을 제외한 후에 세미콜론을 붙인 선언문이며, 컴파일러에게 함수의 형식을 제대로 알려주는 역할을 한다. 컴파일러가 함수의 형식을 알지 못할 경우 기본적으로 반환 타입이 [ ]인 함수라고 가정한다.

01. 함수의 정의와 선언

LAB 8-1 곱셈 함수와 나눗셈 함수

두 수의 곱셈, 나눗셈을 하는 함수를 작성하고, 두 정수를 입력받아서 계산 결과를 출력하는 프로그램을 작성해봅시다(단, 나눗셈의 경우 나누는 수가 0인 경우 0.0을 반환합니다.).

```
두 정수를 공백으로 구분하여 입력하세요.
7 8
7 * 8 = 56
7 / 8 = 0.875000
```

두 정수의 곱셈 함수는 반환 타입도 같은 int라서 특별히 신경 쓸 필요가 없으나 두 정수의 나눗셈 함수의 경우 반환 타입이 double이어야 하고, 나누는 수가 0인 경우 처리와 나눗셈 연산자의 피연산자의 타입 변환도 필요하다. 또한 printf 호출 시 형식지정자는 %f이다.

01. 함수의 정의와 선언

LAB 8-1 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int Multiply(int a, int b)
05  {
06      return a * b;
07  }
08
09  double Divide(int a, int b)
10  {
11      if(!b)
12          return 0.0;
13      else
14          return (double)a / b;
15  }
16
```

01. 함수의 정의와 선언

LAB 8-1 정답

```
17  int main( )
18  {
19      int a, b;
20      printf("두 정수를 공백으로 구분하여 입력하세요.\r\n");
21      scanf("%d%d", &a, &b);
22
23      printf("%d * %d = %d\r\n", a, b, Multiply(a, b));
24      printf("%d / %d = %f", a, b, Divide(a, b));
25  }
```

02

함수 인자

02. 함수 인자

I. 실인자와 매개변수

- **실인자**
  - 함수를 호출할 때 전달되는 값 자체
- **매개변수**
  - 실인자를 받는 변수로서 함수 정의에서 함수 이름 옆의 괄호 안에 선언됨

02. 함수 인자

I. 실인자와 매개변수

[코드 8-13] 실인자와 매개변수

```
01  int Square(int num )
02  {
03     return num * num;
04  }
05
06  int main( )
07  {
08     Square(1);
09
10     int a = 3;
11    Square(a);
12  }
```

02. 함수 인자

II. 값 전달

[코드 8-14] 값 전달

```
01  #include <stdio.h>
02
03  void func(int param)
04  {
05      param = 2;
06  }
07
08  int main( )
09  {
10      int arg = 1;
11
12      printf("Before: %d\r\n", arg);
13      func(arg);
14      printf("After: %d", arg);
15  }
```

```
Before: 1
After: 1
```

02. 함수 인자

II. 값 전달

- 인자로 배열을 사용하는 경우

[코드 8-15] 배열 전달

```
01  #include <stdio.h>
02
03  void func(int param[1])
04  {
05      param[0] = 2;
06  }
07
08  int main( )
09  {
10      int arr[1] = { 1 };
11
12      printf("Before: %d\r\n", arr[0]);
13      func(arr);
14      printf("After: %d", arr[0]);
15  }
```

```
Before: 1
After: 2
```

02. 함수 인자

III. 가변 인자 함수

- **가변 인자 함수의 개념**
  - 가변적인 개수의 실인자를 전달할 수 있는 함수

- **가변 인자 함수의 정의**

```
반환형 함수명(매개변수타입1 매개변수1, ···, 매개변수타입N 매개변수N, ···)
{
    명령절(0개 이상의 명령문 모임)
}
```

하나 더 알기 **매개변수N의 역할**

- 가장 마지막 매개변수N은 함수에 전달된 실인자들 중에서 매개변수 개수를 넘어서 매개변수가 받을 수 없는 실인자를 구하기 위한 기준점이 됨

02. 함수 인자

III. 가변 인자 함수

- 대표적인 가변 인자 함수 printf

```
int printf(const char* format, ···)
{
    명령절(0개 이상 명령문)
}
```

[코드 8-16] 가변 인자 전달

```
01  #include <stdio.h>
02
03  int main( )
04  {
05     printf("%d, %d, %d", 1, 3.14);
06  }
```

```
1, 1374389535, -1498416968
```

02. 함수 인자

확인문제2

1. 실인자와 매개변수의 타입이 서로 다를 경우 실인자는 어떻게 전달되는지 설명하시오.

2. 다음 코드의 실행 결과는 무엇인가?

```
01  #include <stdio.h>
02
03  void func(int arg1, int arg2[1])
04  {
05      arg1 = 1;
06      arg2[0] = 2;
07  }
08
09  int main()
10  {
11      int a = 0;
12      int arr[1] = { 0 };
13      func(a, arr);
14
15      printf("a: %d, arr[0]: %d", a, arr[0]);
16  }
```

02. 함수 인자

3. 함수 func의 반환 타입은 int이다. 다음 코드처럼 printf를 이용하여 func의 반환 값을 출력할 때 잘못된 부분을 찾고 올바르게 수정하시오.

```
printf("%f", func());
```

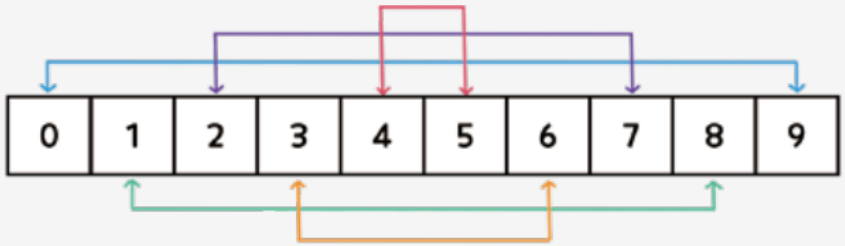
02. 함수 인자

LAB 8-2 배열 요소를 거꾸로 재배치하기

배열 arr이 다음과 같이 정의될 때, arr을 인자로 받아서 배열 요소를 거꾸로 재배치하는 함수 void Reverse(int arr[10])를 구현하고 결과를 출력하는 프로그램을 작성해봅시다.

```
int arr[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

1. Reverse 함수의 매개변수가 배열이라서 참조 전달 방식처럼 동작하므로 함수 안에서 배열의 요소를 거꾸로 재배치할 경우 함수 호출 전 배열 역시 영향을 받는다.
2. 첫 번째 요소와 마지막 요소를 교환하면서 거꾸로 배치를 수행한다. 이때 인덱스는 0부터 4까지 5회만 시행해야 한다.



02. 함수 인자

LAB 7-2 정답

```
01  #include <stdio.h>
02
03  void Reverse(int arr[10])
04  {
05     for(int i = 0; i < 5; i++)
06     {
07        int temp = arr[i];
08        arr[i] = arr[9 - i];
09        arr[9 - i] = temp;
10     }
11  }
12
13  int main( )
14  {
15     int arr[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
16     Reverse(arr);
17     for(int i = 0; i < 10; i++)
18        printf("%d, ", arr[i]);
19  }
```

03

함수 호출

03. 함수 호출

I. 스택 프레임

- **스택 프레임의 개념**
  - 함수 하나가 호출될 때 때 필요한 전용 메모리

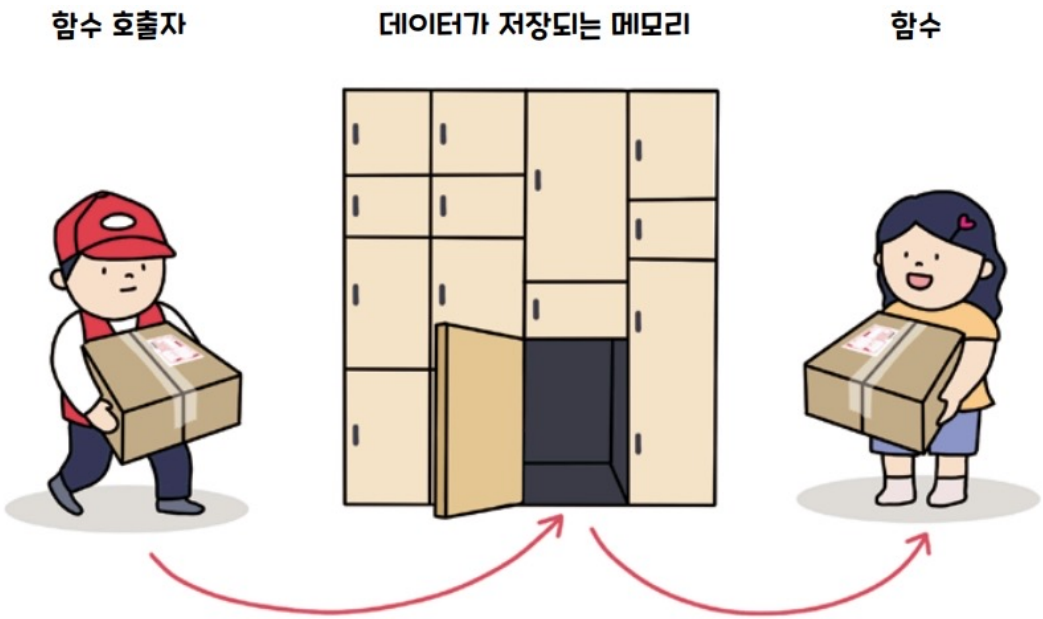


그림 8-3 함수 호출자와 함수, 메모리의 관계

03. 함수 호출

I. 스택 프레임

- 스택 프레임의 개념

[코드 8-17] 스택 프레임

```
01  void funcC(int arg)
02  {
03      arg = 1;
04  }
05
06  void funcB(int arg)
07  {
08      funcC(arg + 1);
09  }
10
11  void funcA(int arg)
12  {
13      funcB(arg + 1);
14  }
15
16  int main( )
17  {
18      funcA(1);
19  }
```

03. 함수 호출

I. 스택 프레임

- 스택 프레임의 개념

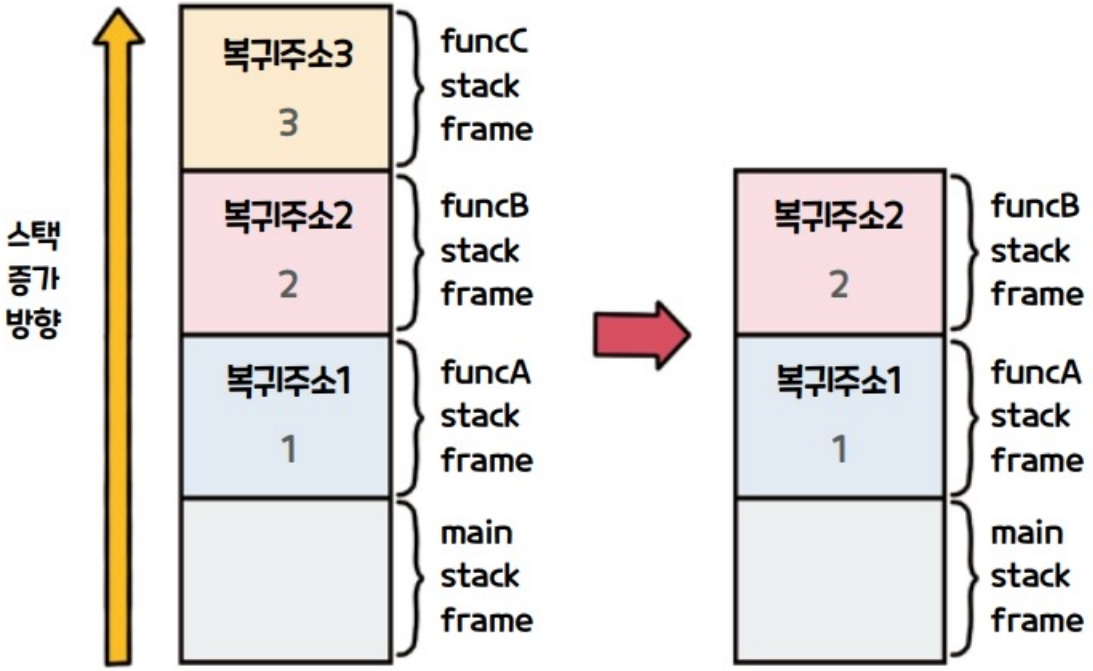


그림 8-4 스택 프레임

03. 함수 호출

I. 스택 프레임

- **스택 프레임의 개념**

하나 더 알기 **LIFO 구조**

- 가장 나중에 쌓인 것이 가장 먼저 빠지는 구조를 후입선출이라 하고, 보통 LIFO(Last In, First Out)라고도 함
- 컴퓨터는 메모리를 관리할 때 LIFO 구조를 많이 사용하는데, 대표적인 것이 바로 스택 프레임이 쌓이는 스택 영역임

03. 함수 호출

II. 재귀 호출

- **재귀 호출의 개념**
  - 자기 자신을 호출하는 것
  - 재귀 호출이 많아질수록 스택 프레임의 층수도 점점 높아지고, 호출이 무한 반복됨
  - 스택 프레임의 전체 크기가 스택 영역의 크기 제한을 넘어서면 컴퓨터는 프로그램을 중지함

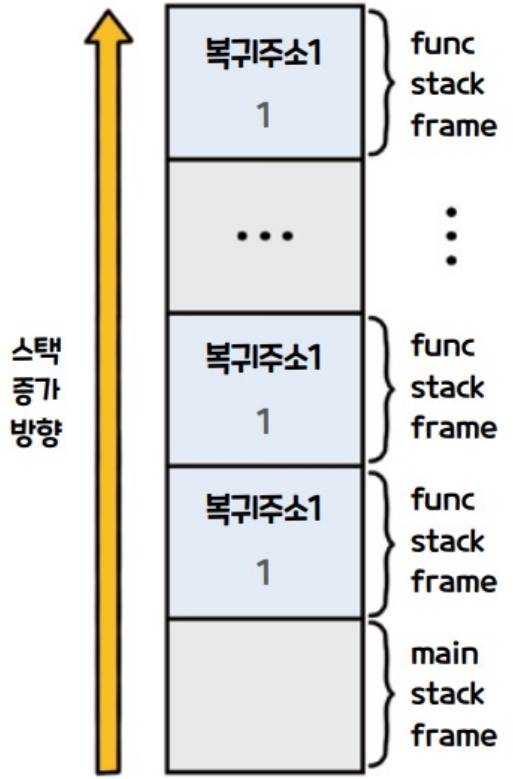


그림 8-5 재귀 호출로 인한 스택 프레임 증가

03. 함수 호출

II. 재귀 호출

- 재귀 호출의 개념

[코드 8-18] 재귀 호출

```
01  void func(int arg)
02  {
03      func(arg);
04  }
05
06  int main( )
07  {
08      func(1);
09  }
```

03. 함수 호출

II. 재귀 호출

- 재귀 호출의 장점

[코드 8-19] 일반 호출과 재귀 호출

```
01  #include <stdio.h>
02
03  int Sum(int N)
04  {
05      int result = 0;
06      for(int i = 1; i <= N; i++)
07      {
08          result += i;
09      }
10
11      return result;
12  }
13
14  int RecursiveSum(int N)
15  {
```

03. 함수 호출

II. 재귀 호출

- 재귀 호출의 장점

[코드 8-19] 일반 호출과 재귀 호출

```
16      if(N == 1)
17          return 1;      // 반환
18
19      return N + RecursiveSum(N - 1); // 재귀 호출
20  }
21
22  int main( )
23  {
24      printf("Sum: %d\r\n", Sum(100));
25      printf("RecursiveSum: %d", RecursiveSum(100));
26  }
```

```
Sum: 5050
RecursiveSum: 5050
```

03. 함수 호출

확인문제3

1. 다음 빈칸에 들어갈 단어를 채우시오.

함수가 호출될 때 함수에 전달하는 실인자와 함수가 반환할 때 되돌아와야 할 복귀주소를 저장할 함수 전용 메모리 영역이 필요하다. 함수 전용 메모리를 [ ]이라고 한다.

2. 다음 빈칸에 들어갈 단어를 채우시오.

함수 안에서 자기 자신 함수를 호출하는 것을 [ ]이라고 한다. [ ]은 무한히 호출이 일어나기 때문에 반드시 특정 조건에서 [ ]을 중단해야 한다. 만일 그렇지 못할 경우 [ ]의 층수가 지나치게 높아져서 프로그램이 중지된다.

03. 함수 호출

LAB 8-3 **재귀 호출을 이용한 n! 구하기**

정수 n을 입력받아서 n!을 출력하는 프로그램을 재귀 호출을 이용하여 작성해봅시다.

재귀 함수인 factorial의 반환 값은 n*factorial(n-1)로 재귀 호출을 한다. 이때 무한 호출을 방지하기 위하여 n이 1일 때는 1을 반환한다.

03. 함수 호출

LAB 8-3 정답

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  int factorial(int n)
05  {
06     if(n == 1)
07        return 1;
08
09     return n * factorial(n - 1);
10  }
11
12  int main( )
13  {
14     int n;
15     printf("정수를 입력하세요.\r\n");
16     scanf("%d", &n);
17
18     printf("n!: %d", factorial(n));
19  }
```

[실전예제]

재귀 호출을 이용한
배열 요소의 합 구하기

[실전예제] 재귀 호출을 이용한 배열 요소의 합 구하기

[문제]

다음과 같은 배열 arr에 대해서 배열 요소의 모든 합을 구해서 출력하는 프로그램을 반복문을 사용하지 않고 재귀 호출을 이용하여 작성하시오.

```
int arr[] = { 3, 5, 1, 10, 9, 8, 2, 6, 4, 7 };
```

실행 결과

```
Sum: 55
```

[실전예제] 재귀 호출을 이용한 배열 요소의 합 구하기

[해결]

1. 함수 Sum의 두 번째 매개변수는 int arr[]로 요소의 개수가 정해지지 않았다. 이런 경우 가변 길이 배열을 인자로 받을 수 있다.

2. 배열이 함수의 인자로 전달되는 경우 참조 전달 방식처럼 동작하므로 재귀 호출을 사용해도 배열 복사의 비용이 발생하지 않는다.

[실전예제] 재귀 호출을 이용한 배열 요소의 합 구하기

[해결]

```
01  #include <stdio.h>
02
03  int Sum(int index, int arr[])
04  {
05     if(index == 0)
06        return arr[0];
07
08     return arr[index] + Sum(index - 1, arr);
09  }
10
11  int main( )
12  {
13     int arr[] = { 3, 5, 1, 10, 9, 8, 2, 6, 4, 7 };
14
15     int count = sizeof(arr) / sizeof(arr[0]);
16     printf("Sum: %d", Sum(count - 1, arr));
17  }
```

Thank you!

한빛아카데미 Hanbit Academy, Inc.