작성자 : 김기희 (cupid072@naver.com)

함수

- 학습목표
  - 함수의 정의와 함수 사용의 장점을 설명할 수 있다.
  - 함수의 선언 방법과 역할에 대해 설명할 수 있다.
  - 함수를 정의하고 호출할 수 있다.
  - 함수의 형식인자(파라미터)와 전달인자(어규먼트)에 대해 설명할 수 있다.
  - 함수 파라미터의 특징을 설명할 수 있다.
  - 가변 인수 파라미터를 사용하여 프로그램을 작성할 수 있다.
  - 함수의 반환형과 return 문에 대해 설명할 수 있다.
  - inline 함수에 대하여 설명할 수 있다.

프리젠테이션 사용 불가

함수 (function)

- 함수(function)
  - 특정한 작업을 수행하도록 독립적으로 작성된 명령 코드 블록

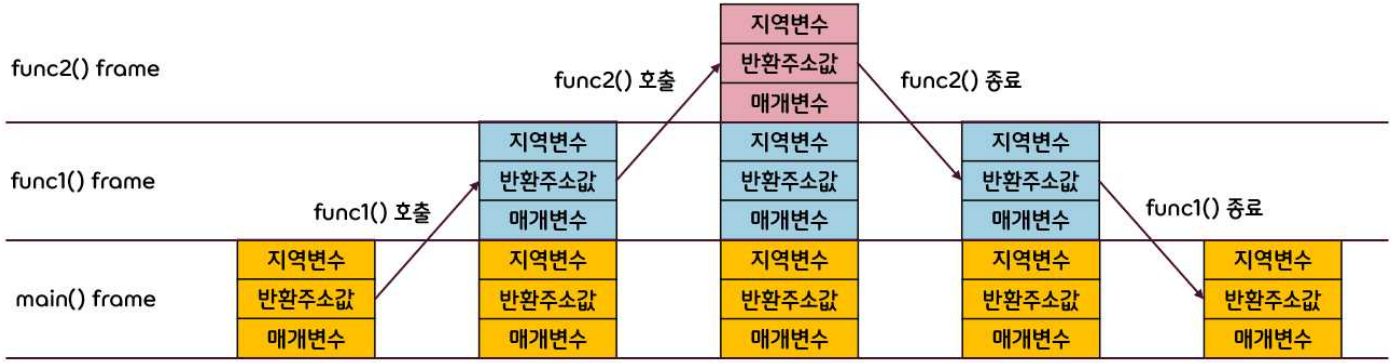
- 함수의 사용 목적
  - 프로그램 내에서 반복적으로 수행되는 부분을 함수로 작성하여 필요 시 마다 호출하여 사용
  - 특정 기능별로 나누어 작성함으로써 구조적인 프로그래밍이 가능하게 한다.

- 함수의 종류
  - library function : 벤더사에서 제공하는 함수
  - user define function : 사용자 정의 함수

프리젠테이션 사용 불가

함수의 실행 원리

- 함수가 호출되면 Stack(Function Call Stack) 세그먼트에 함수 실행을 위한 공간이 할당되고 할당된 공간을 사용하여 함수가 실행된다.
- 함수 실행을 위해 Stack 상에 할당된 공간을 stack frame이라고 한다.
- Stack 세그먼트는 함수의 호출과 관계되는 지역변수(매개변수 포함)가 저장되는 영역이다.
- 더 이상 Stack Frame을 생성할 수 없을 경우 Stack Overflow가 발생한다.



함수의 선언과 정의, 호출

- 함수의 선언

  함수 원형(prototype)을 컴파일러에게 알려주는 역할을 한다.

  ❖ 함수 원형

  함수의 이름, 함수의 반환 타입, 함수의 매개변수(parameter) 타입

- 함수의 정의

  함수가 수행해야 할 실제 기능을 기술

- 함수의 호출

  작성된 함수를 사용하기 위해 함수 이름을 사용하여 호출하는 것

프리젠테이션 사용 불가

함수의 선언

- 함수의 선언
  함수의 선언은 함수 원형을 컴파일러에게 알려주는 역할을 한다.

- 함수 선언 형식
  [extern] 리턴결과형 함수명(매개변수목록);

- 함수 선언의 예
  int sum(int, int);

- 함수의 선언은 함수명과 함께 해당 함수를 실행하기 위해 필요한 데이터(매개변수의 타입)의 수와 데이터의 타입, 순서 그리고 함수의 실행결과로써 반환되는 값의 데이터형을 컴파일러에게 알리는 역할을 한다.
- 함수 선언에 extern 지정자를 지정하지 않을 경우 묵시적으로 extern 지정자가 부여된다.
- 함수는 함수 호출 이전에 선언되어야 한다.
- 함수의 선언은 중복이 가능하지만 권장되지는 않는다.
- 함수 선언의 매개변수는 변수명을 지정하지 않아도 된다.

무단 재배포 및 상업적 사용 불가

함수의 정의

- 함수의 정의는 함수가 수행해야 할 실제 기능을 기술하는 것을 말하다.

```
int sum(int, int);                    // 함수의 선언

int sum(int a, int b) {               // 함수의 정의
    return a + b;
}
```

- 함수의 선언과 달리 함수의 정의는 매개변수에 대한 변수명을 생략할 수 없다.
- 함수는 중복하여 정의할 수 없다.
- 함수는 프로그램 소스 파일 어느 곳에서 든 정의할 수 있다.
- 함수 정의 이전에 함수의 선언이 없을 경우 함수 정의는 함수 선언의 기능도 수행한다.
- 함수 외부에서 정의되는 정적(static) 함수는 파일 범위의 참조범위를 가지므로 함수가 사용되는 파일에 선언과 동시에 정의하는 것이 일반적이다.

무단 배포 및 상업적 이용 불가

함수의 호출

- 함수의 호출은 선언 및 정의된 함수를 사용하기 위해 함수명을 사용하여 호출하는 것을 말한다.

```
int sum(int, int);                  // 함수의 선언
.....
int result = sum(10, 20);           // 함수의 호출
.....
int sum(int a, int b) {             // 함수의 정의
    return a + b;
}
```

- 함수가 호출되면 실행 제어는 호출된 함수의 시작부분으로 이동한다.
- 함수 실행 도중 return 구문을 만나면 return 식의 값을 반환하며 실행제어는 함수 호출 부로 이동한 후 다음 라인을 계속 수행한다.
- 함수 호출 시 함수 호출부에서는 함수가 종료되기 전까지 실행 대기 상태에 들어서게 되며 이러한 상태를 Blocking 상태라고 한다.

프리젠테이션 사용 불가

함수 파라미터(매개변수) #1

- 파라미터는 함수 실행에 필요한 값을 전달받기 위한 변수로써 매개변수라고도 한다.
- 파라미터는 함수 내부에서만 사용 가능한 자동(지역)변수이다.
- 함수 선언부에는 파라미터에 대한 변수명을 기술하지 않아도 된다.
- 함수 정의부에는 파라미터에 대한 변수명을 반드시 기술하여야 한다.
- 함수 선언부와 함수 정의부의 파라미터에 대한 변수명은 반드시 일치하여야 하는 것은 아니다.

```
int function(int, int);                 // 함수 선언부의 파라미터는 변수명의 기술을 생략할 수 있다.
.....
int result = function(10, 20);
.....
int funcion(int a, int b) {             // 함수 정의부의 파라미터는 변수명의 기술을 생략할 수 없다.
.....
}
```

프리젠테이션 사용 불가

함수 파라미터(매개변수) #2

- 함수 파라미터의 자료형은 함수 호출 시 전달되는 argument(전달 인자)의 타입 체크를 위해 필요하다.

- 함수 파라미터를 가지지 않는 함수는 함수 호출 시 전달 인자에 대한 데이터 타입 체크가 지원되지 않는다.

```
void function();
function();             // OK
function(10);           // OK
function(10, 3.14);     // OK
```

- 함수 파라미터를 가지지 않는 함수에 대하여 전달 인자의 타입 체크를 지원 받고자 하는 경우 함수 파라미터를 void 타입으로 기술한다.

```
void function(void);
void();                 // OK
void(10);               // ERROR
```

- void 파라미터는 다른 파라미터와 함께 사용할 없으며 파라미터 이름 또한 가질 수 없다.

프리젠테이션 사용 불가

가변 파라미터(가변 인수) #1

- 가변 파라미터는 함수 호출 시 전달 인자의 수가 정해지지 않았을 때 사용된다.

```
#include <stdarg.h>                    // 가변 파라미터를 사용하기 위한 함수가 선언되어 있는 헤더
void print(int, ...);                  // 가변 파라미터 함수의 선언
int main() {
        print(5, 1, 2, 3, 4, 5);       // 가변 파라미터 함수 호출
        return 0;
}
void print(int count, ...) {           // 가변 파라미터 함수의 정의
        va_list args;                  // 가변 파라미터의 시작 주소 저장을 위한 포인터 변수
        va_start(args, count);         // 가변 파라미터의 시작 주소를 args 에 할당 (매크로 함수)
        for (int i = 0; i < count; i++) {
                printf("%d ", va_arg(args, int));   // args 포인터가 가리키는 곳의 값을 읽어 들인 후 포인터를 다음 인수 위치로 이동시킨다.
        }
        va_end(args);                  // 가변인자 작업 종료 함수로써 반드시 호출해 줄 것을 권장한다.
}
```

프로젝트 외 사용 불가

가변 파라미터(가변 인수) #2

- 가변 파라미터는 다음과 같은 제약이 따른다.
  - 가변 파라미터는 가장 마지막 파라미터로 기술되어야 한다.
  - 하나의 함수에 복 수의 가변 파라미터를 사용할 수 없다.

- 가변 파라미터를 사용하는 함수

```
int printf (const char *template, ...)
int scanf (const char *template, ...)
int fprintf (FILE *stream, const char *template, ...)
int fscanf (FILE *stream, const char *template, ...)
int sprintf (char *s, const char *template, ...)
int sscanf (const char *s, const char *template, ...)
int snprintf (char *s, size_t size, const char *template, ...)
```

프리젠테이션 사용 불가

함수의 반환 형과 return 문

- 함수 선언에서 함수명 앞의 자료형은 함수 실행 결과로 반환되는 데이터의 자료형이다.

```
int function(int, int);                    // function 함수는 int형 두 개의 값을 받으며 실행 결과로 int 형 값을 반환
```

- void 타입이 아닌 자료형을 반환하는 함수는 반드시 명시적인 return 문에 의해 반환 자료형의 값을 반환하여야 한다.

```
int function(int a, int b) {
    ......
    return 식;
}
```

- void 타입을 반환하는 함수는 return 문을 기술하지 않을 수 있다.
- void 타입을 반환하는 함수의 return 문은 식을 가져서는 안 된다. (return 문만 기술)

프리젠테이션 사용 불가

return 문 #1

- return 문의 기능은 다음과 같다.
  - 함수 실행을 종료한다.
  - 함수 실행 종료와 함께 값을 함수 호출 부로 반환한다.

- return 문의 형식

  return [식];

- ※ return문을 뒤 따르는 식은 함수가 반환하는 자료형의 식이어야 한다.

```
int function();          return 10;        // OK
int function();          return 3.14;      // ERROR
double function();       return 10;        // OK  int형이 double 형으로 자동 형변환 되어 반환
void function();         return;           // OK
void function();         return 10;        // ERROR
```

프리젠테이션 사용 불가

return 문 #2

- 함수 내부에서 return 문을 만나면 return 문을 뒤 따르는 식의 값을 반환하며 함수를 종료한다.
- 이것은 return 문 이 후의 문장은 절대 실행 불가능한 코드임을 의미한다.

```
void function() {
    문장1;
    문장2;
    return;                    // return 문에 의해 함수 종료
    문장n;                     // 문장n은 절대 도달할 수 없는 코드
}
```

- 위와 같이 절대 도달할 수 없는 코드를 가리켜 Unreachable Code라고 한다.
- 또한 실행 불가능한 코드라는 의미로 Dead Code라고도 한다.
- 함수 만이 아니라 모든 코드 기술에 있어서 Unreachable Code 또는 Dead Code가 발생하지 않도록 기술해야 한다.
- 함수 실행의 결과 값은 파라미터를 통해 반환할 수도 있다.

프리젠테이션 사용 불가

함수의 호출 기법에 따른 분류

- Call by Name

  전달 인자(argument) 없이 함수의 이름만으로 호출하는 방식

- Call by Value

  함수 호출 시 전달 인자의 값이 복사되어 전달되는 방식

  피 호출 함수의 조작으로부터 전달 인자의 값은 항상 보호된다.

- Call by Pointer(Reference)

  함수 호출 시 전달 인자로써 주소 값을 전달하는 방식

  피 호출 함수내에서는 간접 참조 연산자를 통해 전달 인자의 값을 변경할 수 있다.

코리젠테이션 사용 불가

Call by Name 함수

- Call by Name 함수는 전달 인자 없이 함수의 이름만으로 호출하는 함수를 말한다.

```
int function(void) {
    ......
    return 식;
}
......
int value = function();
```

프리젠테이션 사용 불가

Call by Value 함수

- Call by Value 함수는 전달 인자의 값을 복사하여 사용하는 함수로 주로 기본 자료형 타입을 파라미터로 하는 함수이다.

```
double function(int age, double weight) {
    ......
    return 식;
}
......
int age = 30;
double weight = 65.0;
double average;
average = function(age, weight);          // 함수 호출 시 변수의 값을 전달인자로 사용
average = function(20, 63.5);             // 함수 호출 시 상수 값을 전달인자로 사용
```

프리젠테이션 사용 불가

Call by Pointer(Reference) 함수 #1

- Call by Pointer 함수는 주소 값을 전달 인자로 하는 함수이다.

```
void swap(int* x, int* y) {
        int tmp = *x;
        *x = *y;
        *y = tmp;
}

int a = 10;
int b = 20;
swap(&a, &b);          // OK 변수 a의 주소와 변수 b의 주소가 전달
swap(10, 20);          // ERROR 10번지와 20번지는 어디일까?
```

프리젠테이션 사용 불가

Call by Pointer(Reference) 함수 #2

Call by Name

```
void swap(int x, int y) {
    printf("x : %p\n", &x);
    printf("y : %p\n", &y);
    int tmp = x;
    x = y;
    y = tmp;
}

int main() {
    int a = 10;
    int b = 20;
    printf("a : %p\n", &a);
    printf("b : %p\n", &b);
    swap(a, b);
    return 0;
}
```

Call by Pointer

```
void swap(int* x, int* y) {
    printf("x : %p\n", x);
    printf("y : %p\n", y);
    int tmp = x;
    x = y;
    y = tmp;
}

int main() {
    int a = 10;
    int b = 20;
    printf("a : %p\n", &a);
    printf("b : %p\n", &b);
    swap(&a, &b);
    return 0;
}
```

프리젠테이션 사용 불가

inline 함수

- inline 함수는 C99부터 지원되는 기능으로 inline 지시자를 사용하여 선언된 함수로써 함수를 더 빠르게 실행할 수 있도록 지시한다.
- inline 지시자에 의해 선언된 함수의 호출부는 컴파일 단계에서 함수의 실행 코드로 대치한다.
- inline 함수는 함수의 호출부를 함수 실행 코드로 대치함으로써 실행 속도는 빠르지만 컴파일 후 생성되는 오브젝트 파일의 크기가 증가할 수 있다.
- 컴파일러의 최적화 기능에 의해 inline 지시자를 사용하여 선언한 함수라 할지라도 inlining 여부가 결정된다.

```
static inline void print(int*, int);                    // inline 함수의 선언
                                                        // gcc 에서 inline 함수는 명시적으로 static 또는 extern 지시자를 사용해야 함.
int main(void) {
        int ary[] = { 1, 2, 3, 4, 5 };
        int size = (int)(sizeof(ary) / sizeof(ary[0]));
        print(ary, size);
        return 0;
}

inline void print(int* arr, int size) {                 // inline 함수의 정의
        for (int i = 0; i < size; i++)
                printf("%d\n", arr[i]);
}
```

프리젠테이션 사용 불가

재귀호출 함수

- 재귀호출 함수는 일정 조건을 만족하는 경우 함수가 자신을 호출하는 구조의 함수를 말한다.
- 재귀호출 함수는 반복문에 비해 구조는 간단하지만 함수 호출 시 스택을 사용하므로 오버헤드가 발생한다.
- 특정 지점에서 재귀호출을 종료하지 않을 경우 무한 재귀에 빠진다.
- 재귀함수의 무한 호출은 스택을 소모하고 결국에는 Stack Overflow가 발생하여 프로그램이 비정상 종료된다.

- 재귀함수의 예) n!을 구하는 함수

```
int fact(int n) {
    if (n <= 2) return 1;
    return (n * fact(n-10));
}
```

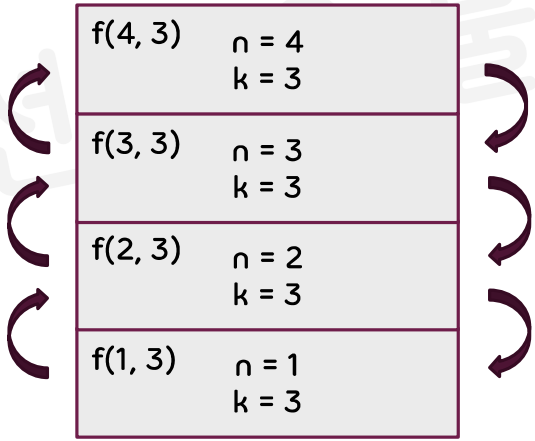
프리젠테이션 사용 불가

재귀호출 함수의 사용 이유

- 재귀호출 함수는 반복문에 비해 구조가 간결하다.
- 반복문을 사용하여 재귀호출 함수의 기능을 구현하려면 스택이나 큐와 같은 자료구조를 함께 사용해야 한다.
- 재귀호출 함수는 변수의 사용과 변수의 상태 변화(side effect)를 최소한으로 제한한다.
- 재귀호출 함수는 함수 호출 전의 상태를 함수 호출 이후까지 유지할 수 있다.

```
f(n, k) {
        if (n > k) return;
        f(n + 1, k);
}

f(1, 3);
```



재귀호출 함수의 구성

- 재귀호출 함수는 다음과 같이 기본부분과 재귀부분으로 구성된다.
  - 기본부분(Base Part) : 재귀호출을 종료하기 위한 부분
  - 재귀부분(Recursive Part) : 부분적 문제 해결을 위한 재귀호출 부분
  - 재귀부분은 재귀호출 이전과 이후에 전반부와 후반부를 가질 수 있다.

```
f(n, k) {
        if (n > k) return;              // 기본부분 (Base Part)
        else {                          // 재귀부분(Recursive Part)
              [전반부]
              f(n+1, k);
              [후반부]
        }
}
```

프리젠테이션 사용 불가

재귀호출 함수 고려사항 #1

- 재귀호출 함수를 구현시에는 다음과 같은 사항을 고려해야 한다.

  - 종료조건 : 기본부분의 구성 방법
  - 처리순서 : 전반부에서 처리할 것인가? 후반부에서 처리할 것인가?
  - 반환값 : 반환 값 산출 및 반환 시점

- 재귀호출 함수는 반드시 재귀호출을 종료할 수 있는 적절한 기본 부분을 가져야만 한다.

```
f(n, k) {
    if (n > k) return;          // 재귀호출 종료를 위한 기본 부분
    f(n + 1, k);
}
```

프린젠테이션 사용 불가

재귀호출 함수 고려사항 #2

- 처리순서는 전반부로 기술하는가? 후반부로 기술하는가에 따라 정순 또는 역순으로 진행된다.

```
f(n, k) {
        if (n > k) return;          // 기본부분
        printf("%d ", n);           // 전반부
        f(n + 1, k);                // 재귀호출부
}
f(1, 3);                            // 결과 1 2 3
```

전반부에서 처리시 정순으로 처리됨

```
f(n, k) {
        if (n > k) return;          // 기본부분
        f(n + 1, k);                // 재귀호출부
        printf("%d ", n);           // 후반부
}
f(1, 3);                            // 결과 3 2 1
```

후반부에서 처리시 역순으로 처리됨

프리젠테이션 사용 불가

재귀호출 함수 고려사항 #3

- 반환 값을 가지는 재귀호출 함수는 기본부분과 재귀호출부분 모든 부분에서 값을 반환해야만 한다.

```
int f(n, k) {
        if (n > k) return 0;              // 기본부분에서 값을 반환
        return n + f(n + 1, k);           // 재귀호출부분에서 값을 반환
}

int sum = f(1, 3);
```

프리젠테이션 사용 불가