

# C 언어 기초 II

작성자 : 홍효상  
이메일 : [hyomee@naver.com](mailto:hyomee@naver.com)  
소스 : [https://github.com/hyomee/c\\_basic](https://github.com/hyomee/c_basic)

# Content

---

## III. C언어 심화

1. 메모리 할당
2. 구조체 확장
3. void 포인터
4. 다차원 포인터
5. 함수 확장
6. 파일 입출력

# 1. 정적 메모리

## “ 데이터를 조장할 메모리 공간은 나누는 작업”

### 메모리 종류

분 류		특 징	프로세스
Stack		자동 변수 지역 변수인 변수가 사용하는 메모리 영역 / 크기가 작고 관리(할당 및 반환)가 자동으로 이루어 짐	스택 세그먼트
Heap		동적 할당할 수 있는 자유 메모리 영역 개발자 스스로 관리(수동) - 제일 큰 영역 할당 및 해제를 신경 써야 함	스택 세그먼트
PE image (실행영역)	Text section	C언어의 소스코드가 번역된 기계어가 저장된 메모리 영역 기본적으로 읽기전용 메모리	코드 세그먼트
	Data section(Read only)	상수 형태를 기술하는 문자열(예: "Hello")이 저장된 메모리 영역	데이터 세그먼트
	Data section(Read/Write)	정적변수나 전역변수들이 사용하는 메모리 영역 별도로 초기화하지 않아도 0으로 초기화(전역변수), Static 전역변수 ( 초기화 되지 않음 ) 관리는 자동이기 때문에 Heap 영역 메모리처럼 할당 및 해제를 신경 쓸 필요 없음	데이터 세그먼트

### 정적 메모리 할당

- 컴파일러가 코드를 기계어로 번역 하는 시점에 변수를 저장 메모리 위치를 배정 하는 것
- 실행 중에 메모리를 변경 할 수 없음

전역 변수

데이터 세그먼트: 프로그램 시작 시 생성 되고 종료 시 소멸

지역 변수

스택 세그먼트: 함수 호출 시점에 할당 되고 함수 종료 시 해제

# 1. 메모리 할당

## 3. 언어 심화 1-1. 메모리 할당

```
#include <stdio.h>
#include <stdlib.h>

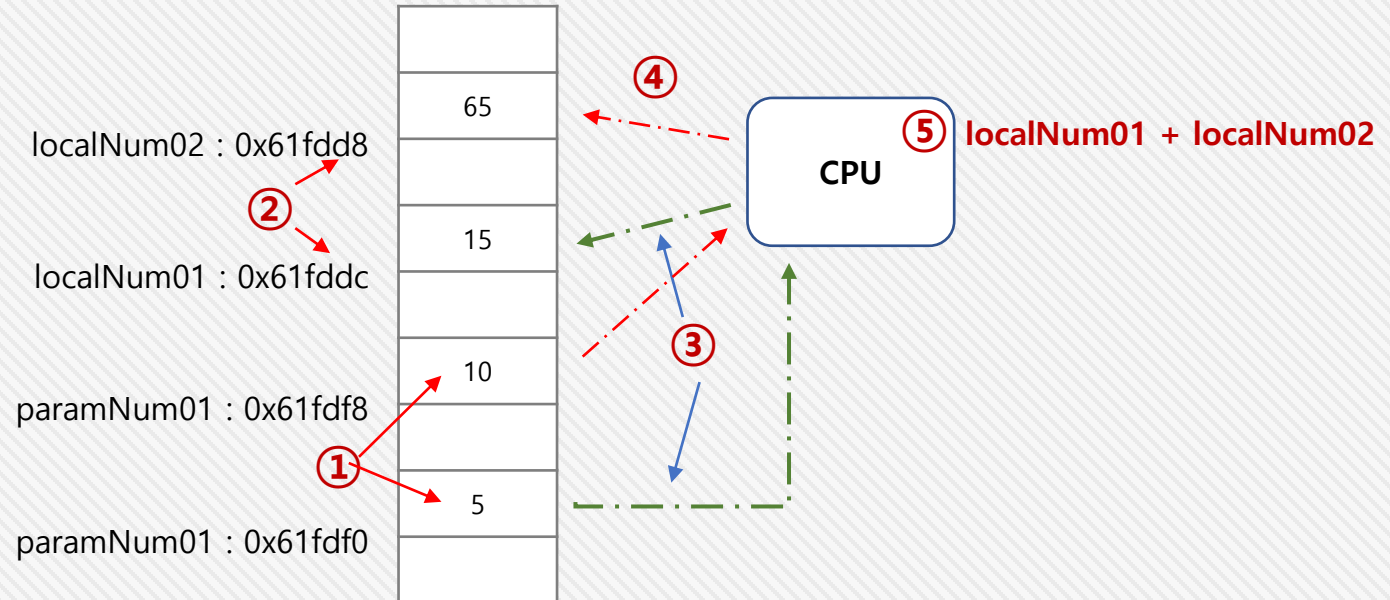
extern int funStack(int paramNum01, int paramNum02);

int main(void) {
    puts("!!! 메모리 할당 !!!");
    int result = funStack(5, 10);
    printf("%d", result);
    return EXIT_SUCCESS;
}
```

```
int funStack(int paramNum01, int paramNum02) { ①
    ② int localNum01, localNum02;

    ③ localNum01 = paramNum01 + 10;
    ④ localNum02 = paramNum02 + 55;

    return localNum01 + localNum02; ⑤
}
```



# 2. 동적 메모리

## 3. 언어 심화 1-1. 메모리 할당

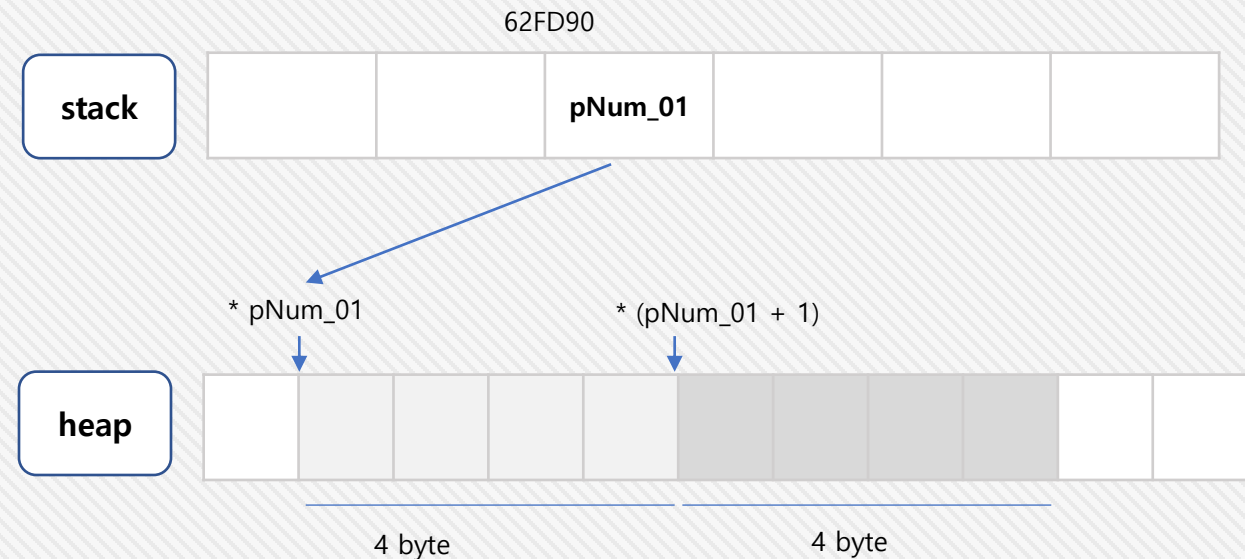
### 정적 메모리 한계

- 정적 메모리는 1M 를 넘을 수 없으므로 그 이상의 데이터를 사용 하기 위해서 **heap** 영역을 사용 해야 함 -> 동적 할당

### 동적 메모리

- 1G 단위 까지 할당
- 개발자가 할당 해야 하고 해제 해야 함
- malloc ( Memory Allocation )
  - 함수 원형: `void *malloc(size_t size) : size_t -> unsigned int` 형
  - 해제 : `free()`

```
void funStaticDynamicMemory(void) {  
    printf("=== 정적 할당 .. 동적 할당 \n");  
    int num = 10;  
    int *pNum = &num;  
    int *pDNum = (int *) malloc(sizeof(int));  
    int *pDNum_01 = (int *) malloc(sizeof(int)*2);  
  
    printf("pNum 주소      :: %p\n", &pNum);  
    printf("pDNum 주소      :: %p\n", &pDNum);  
    printf("pDNum_01 주소      :: %p\n", &pDNum_01);  
    free(pDNum);  
    free(pDNum_01);  
}
```



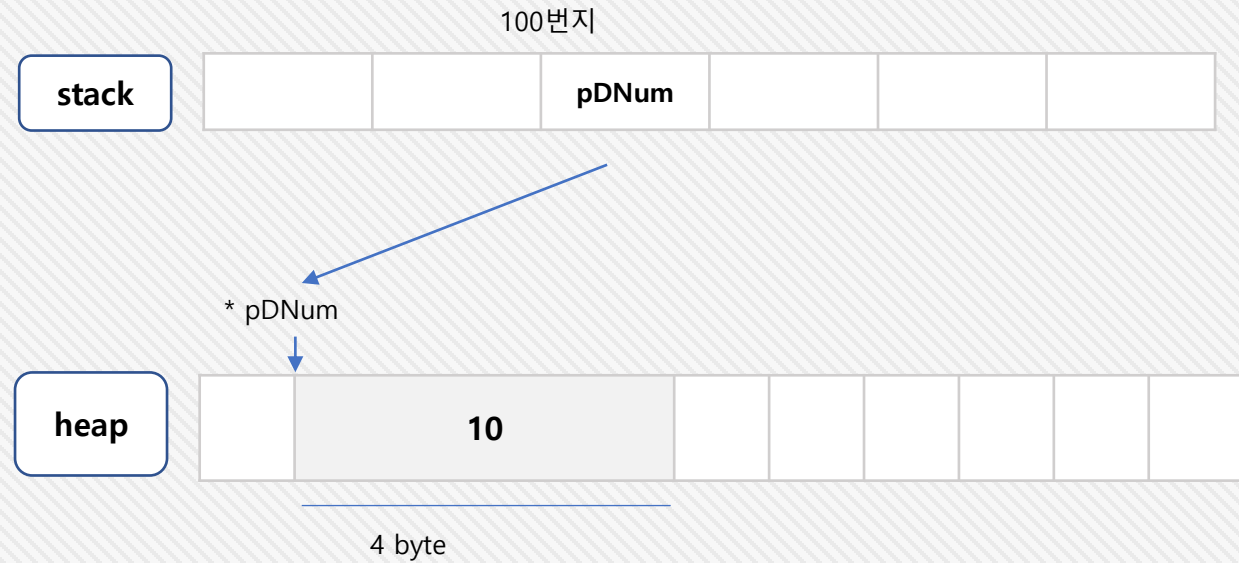
## 2. 동적 메모리

### 3. 언어 심화 1-1. 메모리 할당

#### 예제

```
int *pDNum = (int *) malloc(sizeof(int));
```

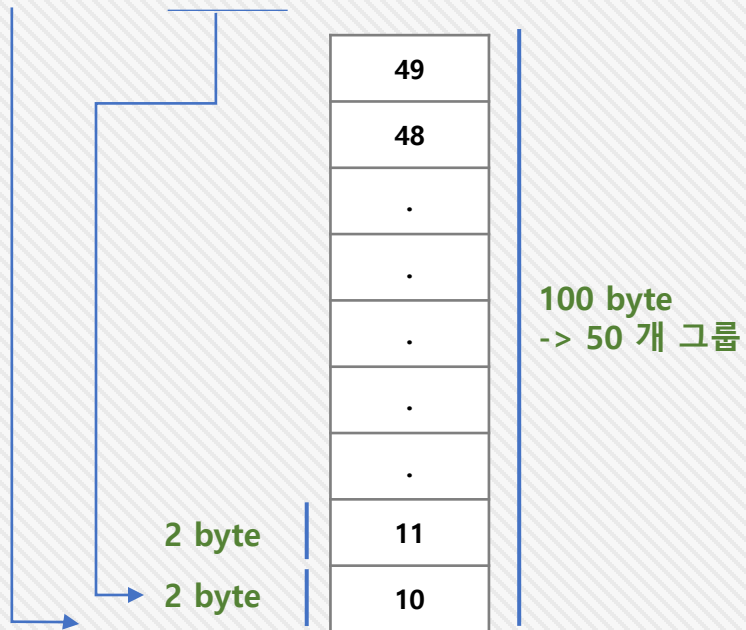
```
void funSetDynamicMemory(void) {  
    printf("=== 메모리 저장 \n");  
    int *pDNum = (int *) malloc(sizeof(int));  
    *pDNum = 10;  
  
    printf("메모리 저장 pDNum :: %d\n", *pDNum);  
    free(pDNum);  
}
```



## 2. 동적 메모리

### 예제

```
short *ptr = (short *) malloc(100);
```



```
void funMolloc(void) {  
    printf("동적 할당 \n");  
    int index;  
    short *ptr = (short *) malloc(100);  
  
    for ( index = 0; index < 50; index++) {  
        *(ptr + index) = index + 10;  
    }  
  
    printf("0x%X\n", ptr);           // 0xB97650  
    printf("%d\n", *(ptr + 1));     // 11  
    printf("%d\n", *(ptr + 49));    // 59  
    printf("%d\n", *(ptr + 50));    // 0  
    free(ptr);  
}
```

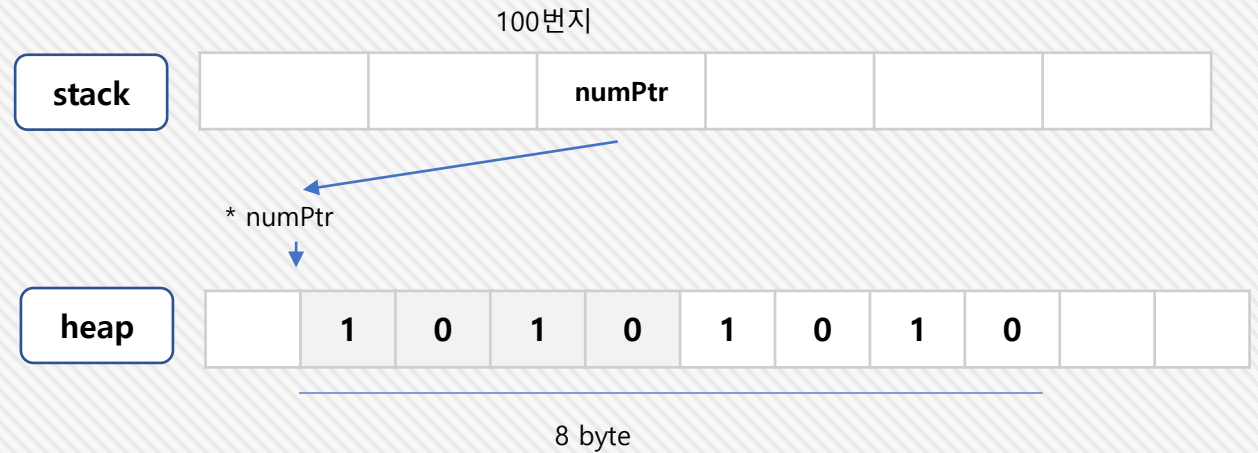
## 2. 동적 메모리

### 3. 언어 심화 1-1. 메모리 할당

#### 예제: 메모리 내용을 한번에 설정

```
#include <stdlib.h>
#include <string.h>

void funSetAllMemory() {
    printf("동적 할당 - 전체 초기화 \n");
    int *numPtr = (int *) malloc(sizeof(int)*2);
    memset(numPtr, 0x10, sizeof(int)*2);
    printf("0x%X\n", *numPtr);    // 0x10101010
    free(numPtr);
}
```





# 1. 배열 속에 구조체

## 배열 속에 구조체

- 배열을 선언 하고 각 요소에 구조체를 할당 한다.

```
typedef struct Student {  
    char name[20];  
    int age;  
    char address[200];  
} STUDENT ;
```

Student 구조체를 STUDENT TYPE으로 재정의

```
void funStructArray() {  
    int index ;  
    STUDENT student[2];  
    student[0] = (STUDENT) {"홍길동", 19, "서울시 송파구 방이동" };  
    student[1] = (STUDENT) {"홍당무", 18, "서울시 송파구 천호동" };  
  
    printf("\n==== 배열 속에 구조체 =====\n");  
    for ( index = 0 ; index < 2; index ++ ) {  
  
        printf("   이름   : %s\n", student[index].name);  
        printf("   나이   : %d\n", student[index].age);  
        printf("   주소   : %s\n", student[index].address);  
    }  
}
```

```
==== 배열 속에 구조체 =====  
이름   : 홍길동  
나이   : 19  
주소   : 서울시 송파구 방이동  
이름   : 홍당무  
나이   : 18  
주소   : 서울시 송파구 천호동
```

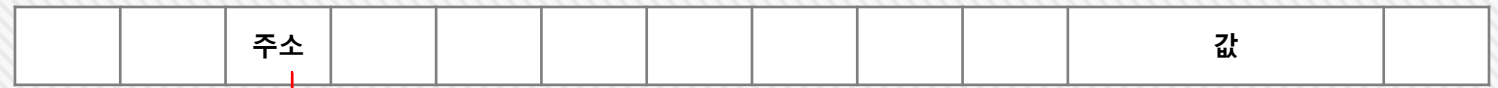
# 1. 구조체 Pointer

## 구조체 포인터 선언

- struct 구조체이름 \*포인터이름 = malloc(sizeof(struct 구조체 이름));
- 멤버 접근 : 포인터이름 -> 멤버명, (\*포인터이름).멤버명

```
struct Customer {  
    char name[20];  
    int age;  
    char address[200];  
};
```

Customer 선언



customer

Customer  
구조체

```
void funStructPointerMallocSingle() {  
    // 구조체 Pointer 선언 , 메모리 할당  
    struct Customer *customer = malloc(sizeof(struct Customer));  
  
    printf("\n==== 구조체 메모리 할당 단일 =====\n");  
    // 데이터 설정  
    strcpy(customer->name, "홍길동");  
    customer->age = 30;  
    strcpy((*(customer)).address, "서울시 송파구 방이동");  
  
    // 출력  
    printf("이름 : %s\n", customer->name);  
    printf("나이 : %d\n", customer->age);  
    printf("주소 : %s\n", (*(customer)).address);  
  
    free(customer);  
}
```

customer 포인터 변수 선언 , 크기 할당

Customer 참조 주소 0번째 Customer구조체 멤버 name

Customer 참조 주소에 있는 값 Customer구조체 멤버 address

==== 구조체 메모리 할당 단일 =====

이름 : 홍길동

나이 : 30

주소 : 서울시 송파구 방이동

# 1. 구조체 Pointer

## 3. 언어 심화 2-1. 구조체 ( 연결 리스트 )

### 구조체 포인터 선언

```
struct Customer {  
    char name[20];  
    int age;  
    char address[200];  
};
```

```
void funStructPointerMallocMulti() {  
    int index;  
    int customerCount = 3;  
  
    // 구조체 Pointer 선언 , 메모리 할당  
    struct Customer *customer = malloc(sizeof(struct Customer)*customerCount);  
  
    printf("\n===== 구조체 메모리 할당 복수 =====\n");  
  
    // 데이터 설정  
    strcpy(customer->name, "홍길동");  
    customer->age = 30;  
    strcpy((*customer).address, "서울시 송파구 방이동");  
  
    strcpy((customer+1)->name, "홍당동");  
    (customer+1)->age = 20;  
    strcpy((*customer+1).address, "서울시 강동구 천호동");  
  
    *(customer+2) = (struct Customer) {"바독이", 20, "서울시 강동구 길동"} ;  
  
    // 출력  
    for ( index = 0 ; index < customerCount; index ++ ) {  
        printf("이름 : %s\n", (customer+1)->name);  
        printf("나이 : %d\n", (customer+1)->age);  
        printf("주소 : %s\n", (*(customer+index)).address);  
    }  
  
    free(customer);  
}
```

customer  
pointer 변수  
customer 포인터 변수 선언 , 크기 할당

customer 참조 주소 0번째 Customer 구조체 멤버에 각각 설정

customer 참조 주소 1번째 Customer 구조체 멤버에 각각 설정

customer 참조 주소 2번째 Customer 구조체 멤버에 각각 설정

===== 구조체 메모리 할당 복수 =====

이름 : 홍당동  
나이 : 20  
주소 : 서울시 송파구 방이동  
이름 : 홍당동  
나이 : 20  
주소 : 서울시 강동구 천호동  
이름 : 홍당동  
나이 : 20  
주소 : 서울시 강동구 길동

# 1. 구조체 안에 구조체

## 3. 언어 심화 2-1. 구조체 확장

### 구조체 안에 구조체

```
typedef struct Student {  
    char name[20];  
    int age;  
    char address[200];  
} STUDENT ;  
  
typedef struct Class {  
    char name[20];  
    STUDENT student;  
} CLASS ;  
  
typedef struct School {  
    char name[20];  
    CLASS class;  
} SCHOOL ;
```

```
void funStructStruct() {  
  
    SCHOOL school;  
    strcpy(school.name, "서울학교");  
    strcpy(school.class.name, "1학년");  
    strcpy(school.class.student.name, "홍길동");  
    school.class.student.age = 19;  
    strcpy(school.class.student.address, "서울시 송파구 방이동");  
  
    printf("\n===== 구조체 > 구조체 > 구조체 =====\n");  
    printf("학교명      : %s\n", school.name);  
    printf(" 학년       : %s\n", school.class.name);  
    printf("  이름      : %s\n", school.class.student.name);  
    printf("  나이      : %d\n", school.class.student.age);  
    printf("  주소      : %s\n", school.class.student.address);  
  
}
```

학교명  
학교  
이름  
나이  
주소

```
===== 구조체 > 구조체 > 구조체 =====  
학교명      : 서울학교  
학년       : 1학년  
  이름      : 홍길동  
  나이      : 19  
  주소      : 서울시 송파구 방이동
```

# 1. 구조체 반환

## 3. 언어 심화 2-1. 구조체 확장

### 구조체 반환

- 구조체의 반환은 void pointer로 한다.

```
void funStructReturnMain(void) {  
    STUDENT student = *(STUDENT *)funStructReturn();  
    printf("\n==== 구조체 반환 =====\n");  
  
    printf(" 이름      : %s\n", student.name);  
    printf(" 주소      : %s\n", student.address);  
    printf(" 나이      : %d\n", student.age);  
}
```

```
void *funStructReturn(void) {  
    STUDENT *student = malloc(sizeof(STUDENT));  
    void *vStudentPtr;  
  
    student->age = 19;  
    strcpy(student->address, "서울시 송파구 방이동");  
    strcpy(student->name, "홍길동");  
  
    vStudentPtr = student;  
    free(student);  
    return vStudentPtr;  
}
```

# 1. void 포인터

## 3. 언어 심화 3-1. void 포인터

### Void 포인터

#### void \*포인터 이름

- 대상의 크기가 정해져 있지 않는 pointer
- 모든 자료형을 설정 할 수 있음
- 시작 주소만 있고 끝 주소를 모를 때 사용
- 사용할 크기를 반드시 표기 해야 한다 => 형 변환 문법 사용

```
void main() {  
    int num = 10;  
    void *voidPtrNum = &num;  
    *voidPtrNum = 20;  
}
```

```
26_VoidPointer.c:6:5: warning: dereferencing 'void *' pointer  
    *voidPtrNum = 20;  
    ^~~~~~  
26_VoidPointer.c:6:17: error: invalid use of void expression  
    *voidPtrNum = 20;  
    ^
```

```
void main() {  
    int num = 10;  
    void *voidPtrNum = &num;  
    *(int *)voidPtrNum = 20;  
}
```

- 대상 메모리의 크기를 조절

```
int num = 10;           // 정수형 변수 선언 및 초기화  
char chA = 'a';         // 문자 변수 선언 및 초기화  
  
void *voidPtr;          // void 포인터 변수 선언  
printf("==== 자유로운 형 참조 =====\n");  
voidPtr = &num;          // 정수형 변수 num의 주소를 void 포인터를 저장  
printf("void Point : %d\n", *(int *) voidPtr); // void 포인터 역 참조 하면 에러 이므로 int형으로 변환  
  
voidPtr = &chA;          // 문자 변수 chA의 주소를 void 포인터를 저장  
printf("void Point : %c\n", *(char *) voidPtr); // void 포인터 역 참조 하면 에러 이므로 char형으로 변환
```

```
==== 자유로운 형 참조 =====  
void Point : 10  
void Point : a
```

# 1. void 포인터

## 3. 언어 심화 3-1. void 포인터

### Void 포인터 함수 파라미터, 반환

```
void voidPointerParamReturn(void) {  
    int num = 100;  
    char ch = 'a';  
  
    printf("==== void 포인터 | 파라미터, 반환 =====\n");  
    printf("문자 변환 : %c\n", *((char *) funConvertData(&ch, 'c')));  
    printf("정수 변환 : %d\n", *((char *) funConvertData(&num, 'i')));  
    printf("실수 변환 : %.1f\n", *((float *) funConvertData(&num, 'f')));  
}  
  
void *funConvertData(void *ptr, char type) {  
    void *rnVoidPtr;  
  
    if ( type == 'c' ) {  
        *((char *) ptr) += 1;  
        rnVoidPtr = (char *) ptr;  
    } else if ( type == 'i' ) {  
        *((int *) ptr) += 1;  
        rnVoidPtr = (int *) ptr;  
    } else {  
        *((float *) ptr) += 2.5;  
        rnVoidPtr = (float *) ptr;  
    }  
  
    return rnVoidPtr;  
}
```

사용 시점에 형 변환

사용 시점에 형 변환

```
==== void 포인터 | 자유로운 형 변환 ====  
void Point : 10  
void Point : a  
==== void 포인터 | 파라미터, 반환 =====  
문자 변환 : b  
정수 변환 : 101  
실수 변환 : 2.5
```

# 1. 다차원 포인터

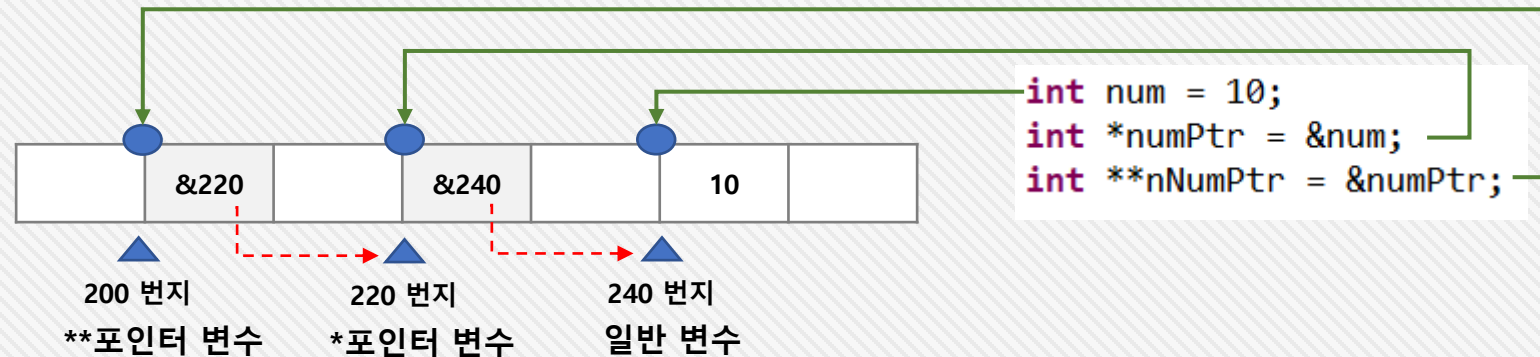
## 다차원 포인터

- 포인터를 여러 개 연결 하여서 n차원을 만드는 것으로 7개 까지 가능 한다.

1차원 포인터



2차원 포인터

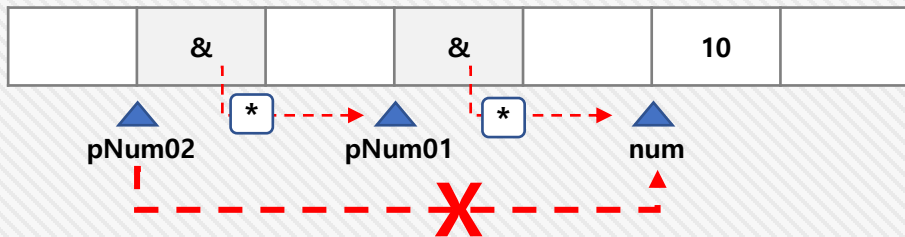




# 1. 다차원 포인터

## 3. 언어 심화 4-1. 다차원 포인터

### 1차원 포인터 + 1차원 포인터



```
int num = 10;
int *pNum01 = &num;
int *pNum02 = &pNum01;
```

```
printf("\n=== 1차원 포인터 + 1차원 포인터 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", *pNum02);
```

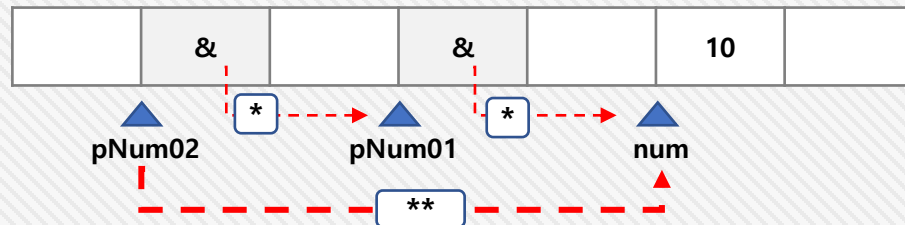
```
num = 20;
printf("\n=== num 20으로 변경 후 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", *pNum02);
```

2차원 포인터와 구조는 같으나  
2차원 포인터 변수가 직접 일반  
변수를 차지 할 수 없다.

```
=== 1차원 포인터 + 1차원 포인터 ===
num : 0xA
pNum01 : 0xA
pNum02 : 0x62FDD8

=== num 20으로 변경 후 ===
num : 0x14
pNum01 : 0x14
pNum02 : 0x62FDD8
```

### 2차원 포인터



```
int num = 10;
int *pNum01 = &num;
int **pNum02 = &pNum01;
```

```
printf("\n=== 2차원 포인터 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", **pNum02);
```

```
num = 20;
printf("\n=== num 20으로 변경 후 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", **pNum02);
```

2차원 포인터로 일반 변수 직접  
참조

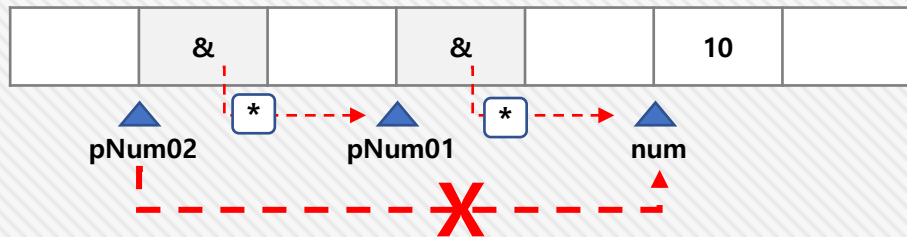
```
=== 2차원 포인터 ===
num : 0xA
pNum01 : 0xA
pNum02 : 0xA

=== num 20으로 변경 후 ===
num : 0x14
pNum01 : 0x14
pNum02 : 0x14
```

# 1. 다차원 포인터

## 3. 언어 심화 4-1. 다차원 포인터

### 2차원 포인터 동적 메모리 할당



```
int num = 10;
int *pNum01 = &num;
int *pNum02 = &pNum01;
```

```
printf("\n=== 1차원 포인터 + 1차원 포인터 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", *pNum02);
```

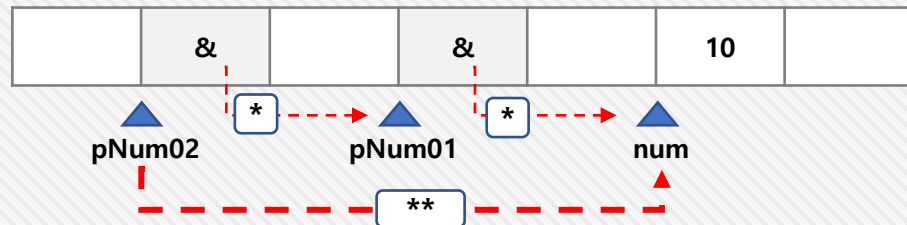
```
num = 20;
printf("\n=== num 20으로 변경 후 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", *pNum02);
```

2차원 포인터와 구조는 같으나  
2차원 포인터 변수가 직접 일반  
변수를 차지 할 수 없다.

```
=== 1차원 포인터 + 1차원 포인터 ===
num : 0xA
pNum01 : 0xA
pNum02 : 0x62FDD8
```

```
=== num 20으로 변경 후 ===
num : 0x14
pNum01 : 0x14
pNum02 : 0x62FDD8
```

### 2차원 포인터



```
int num = 10;
int *pNum01 = &num;
int **pNum02 = &pNum01;
```

```
printf("\n=== 2차원 포인터 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", **pNum02);
```

```
num = 20;
printf("\n=== num 20으로 변경 후 ===\n");
printf("num : 0x%X\n", num);
printf("pNum01 : 0x%X\n", *pNum01);
printf("pNum02 : 0x%X\n", **pNum02);
```

2차원 포인터로 일반 변수 직접  
참조

```
=== 2차원 포인터 ===
num : 0xA
pNum01 : 0xA
pNum02 : 0xA
```

```
=== num 20으로 변경 후 ===
num : 0x14
pNum01 : 0x14
pNum02 : 0x14
```

## 가변 인자 함수

- 함수 생성시 고정 파라미터를 생성 하였는데 파라미터를 가변적으로 만들기 위해서는 고정 매개변수 뒤에 ...을 사용 해서 만들
- 선언: **반환 자료형 함수이름(자료형 고정매개변수, ...)**
- **stdarg.h** 를 사용 하여 가변 인자 함수를 선언 할 수 있다. ( pointer )

```
int main(void) {  
    printf("함수확장 !!!\n");  
    functionDynamicArgument(1, 10);  
    functionDynamicArgument(2, 10, 20);  
    return EXIT_SUCCESS;  
}  
  
void functionDynamicArgument(int args, ...) {  
    printf("%d, ", args);  
}
```

```
함수확장 !!!  
1, 2,
```

## 2. 함수 포인터

### 함수 포인터

- 함수를 배열 또는 구조체에 넣거나, 함수 자체를 함수의 매개 변수로 넘겨 주고, 반환 값을 사용 할 때 이용
- **CallBack 함수**

#### ➤ 함수 주소 출력

```
void function01() {  
    printf("난 함수 ....\n");  
}  
  
void functionPointer() {  
    printf("\n함수 주소 function01() :: %p\n", function01);  
    function01();  
}
```

함수 주소 function01() :: 00000000004015CA  
난 함수 ....

#### ➤ 함수 포인터 선언: 반환자료형 (\*함수포인터이름)()

```
void function01() {  
    printf("난 함수 ....\n");  
}  
  
void funPointerBase() {  
    // 함수포인터 선언  
    void (*funNum)();  
  
    // 함수 포인터에 함수 할당  
    funNum = function01;  
    printf("\n함수 포인터 실행\n");  
    funNum();  
}
```

함수 포인터 실행  
난 함수 ....

## 2. 함수 포인터

### 3. 언어 심화 5-1. 함수 확장

#### ➤ 함수 포인트 선언:

반환자료형 (\*함수포인터이름)(매개변수자료형, 매개변수자료형)

```
int funPointerTest(int a, int b) {  
    return a + b;  
}  
  
void funPointerExp() {  
  
    int num ;  
    // 함수포인터 선언  
    int (*funAdd)(int, int);  
    // 함수 포인터에 함수 할당  
    funAdd = funPointerTest;  
    // 함수 포인터 실행  
    num = funAdd(5, 6);  
  
    printf("\nfunAdd : %d\n", num);  
}
```

funAdd : 11

#### ➤ Callback

```
void funPointerBase() {  
    // 함수포인터 선언  
    void (*funNum)();  
  
    // 함수 포인터에 함수 할당  
    funNum = function01;  
    printf("\n함수 포인터 실행\n");  
    funNum();  
}  
  
int funPointerTest(int a, int b) {  
    return a + b;  
}  
  
int funPointerExpCall(void (*fnAdd)(int, int)) {  
    funPointerBase();  
    return fnAdd;  
}  
  
void funPointerExpCallback() {  
  
    // 함수 포인터 실행  
    int (*funAdd)(int, int) = &funPointerTest;  
    int num = funPointerExpCall(funAdd(12,6));  
  
    printf("\nfunAdd : %d\n", num);  
}
```

함수 포인터 실행  
난 함수 ....

funAdd : 18

# 1. 파일 입출력

3. 언어 심화  
6-1. 파일 입출력

