다른 자료형과 마찬가지로 구조체도 포인터를 선언할 수 있으며 구조체 포인터에는 malloc 함수를 사용하여 동적 메모리를 할당할 수 있습니다.

* **struct 구조체이름 \*포인터이름 = malloc(sizeof(struct 구조체이름));**

***// struct\_alloc\_memory.c***

#define \_CRT\_SECURE\_NO\_WARNINGS // strcpy 보안 경고로 인한 컴파일 에러 방지

#include <stdio.h>

#include <string.h> // strcpy 함수가 선언된 헤더 파일

#include <stdlib.h> // malloc, free 함수가 선언된 헤더 파일

struct Person { // 구조체 정의

char name[20]; // 구조체 멤버 1

int age; // 구조체 멤버 2

char address[100]; // 구조체 멤버 3

};

int main()

{

struct Person \*p1 = malloc(sizeof(struct Person)); // 구조체 포인터 선언, 메모리 할당

// 화살표 연산자로 구조체 멤버에 접근하여 값 할당

strcpy(p1->name, "홍길동");

p1->age = 30;

strcpy(p1->address, "서울시 용산구 한남동");

// 화살표 연산자로 구조체 멤버에 접근하여 값 출력

printf("이름: %s\n", p1->name); // 홍길동

printf("나이: %d\n", p1->age); // 30

printf("주소: %s\n", p1->address); // 서울시 용산구 한남동

free(p1); // 동적 메모리 해제

return 0;

}

***실행 결과***

이름: 홍길동

나이: 30

주소: 서울시 용산구 한남동

먼저 struct Person \*p1과 같이 struct 키워드와 구조체 이름을 사용하여 구조체 포인터를 선언합니다. 이때 일반 변수가 아닌 포인터 변수이므로 반드시 \*을 붙입니다. 그리고 malloc 함수로 메모리를 할당할 때 크기를 알아야 하므로 sizeof(struct Person)과 같이 구조체 크기를 구하여 넣어줍니다.

struct Person **\*p1 = malloc(sizeof(struct Person));**

// 구조체 포인터 선언, 메모리 할당

다소 문법이 복잡하지만 구조체 이름 앞에는 반드시 struct 키워드를 붙여야 한다는 점만 기억하면 쉽습니다. 즉, 포인터를 선언할 때도, sizeof로 크기를 구할 때도 struct 키워드를 넣어줍니다.

그런데 구조체의 멤버에 접근하는 방법이 조금 특이합니다. 지금까지 . (점)을 사용해서 멤버에 접근했지만 **구조체 포인터의 멤버에 접근할 때는 -> (화살표 연산자)를 사용**합니다.

// 화살표 연산자로 구조체 멤버에 접근하여 값 할당

strcpy(**p1->name**, "홍길동");

**p1->age** = 30;

strcpy(**p1->address**, "서울시 용산구 한남동");

// 화살표 연산자로 구조체 멤버에 접근하여 값 출력

printf("이름: %s\n", **p1->name**); // 홍길동

printf("나이: %d\n", **p1->age**); // 30

printf("주소: %s\n", **p1->address**); // 서울시 용산구 한남동

p1->age = 30;과 같이 구조체 포인터의 멤버에 접근한 뒤 값을 할당하고, p1->age와 같이 값을 가져옵니다. p1->name 등의 문자열 멤버는 = (할당 연산자)로 저장할 수 없으므로 strcpy 함수를 사용하면 됩니다.

마지막으로 free(p1);처럼 할당한 메모리를 해제해줍니다. 즉, 구조체에 메모리를 할당할 때는 malloc → 사용 → free 패턴을 기억하세요.

# 구조체 포인터에서 .으로 멤버에 접근하기

구조체 포인터에서 멤버에 접근하려면 p1->age와 같이 화살표 연산자를 사용하는데 괄호와 역참조를 사용하면 . (점)으로 멤버에 접근할 수 있습니다.

p1->age; // 화살표 연산자로 멤버에 접근

(\*p1).age; // 구조체 포인터를 역참조한 뒤 .으로 멤버에 접근

(\*p1).age와 같이 구조체 포인터를 역참조하면 pointer to struct Person에서 pointer to가 제거되어 struct Person이 됩니다. 따라서 .으로 멤버에 접근할 수 있다.

# 구조체의 멤버가 포인터일 때 역참조하기

구조체의 멤버가 포인터일 때 역참조를 하려면 맨 앞에 \*를 붙이면 됩니다. 이때 구조체 변수 앞에 \*가 붙어있더라도 멤버의 역참조이지 구조체 변수의 역참조가 아닙니다.

* **\*구조체변수.멤버**
* **\*구조체포인터->멤버**

***//struct\_member\_dereference.c***

#include <stdio.h>

#include <stdlib.h>

struct Data {

char c1;

int \*numPtr; // 포인터

};

int main()

{

int num1 = 10;

struct Data d1; // 구조체 변수

struct Data \*d2 = malloc(sizeof(struct Data)); // 구조체 포인터에 메모리 할당

d1.numPtr = &num1;

d2->numPtr = &num1;

printf("%d\n", \*d1.numPtr); // 10: 구조체의 멤버를 역참조

printf("%d\n", \*d2->numPtr); // 10: 구조체 포인터의 멤버를 역참조

d2->c1 = 'a';

printf("%c\n", (\*d2).c1); // a: 구조체 포인터를 역참조하여 c1에 접근

// d2->c1과 같음

printf("%d\n", \*(\*d2).numPtr); // 10: 구조체 포인터를 역참조하여 numPtr에 접근한 뒤 다시 역참조

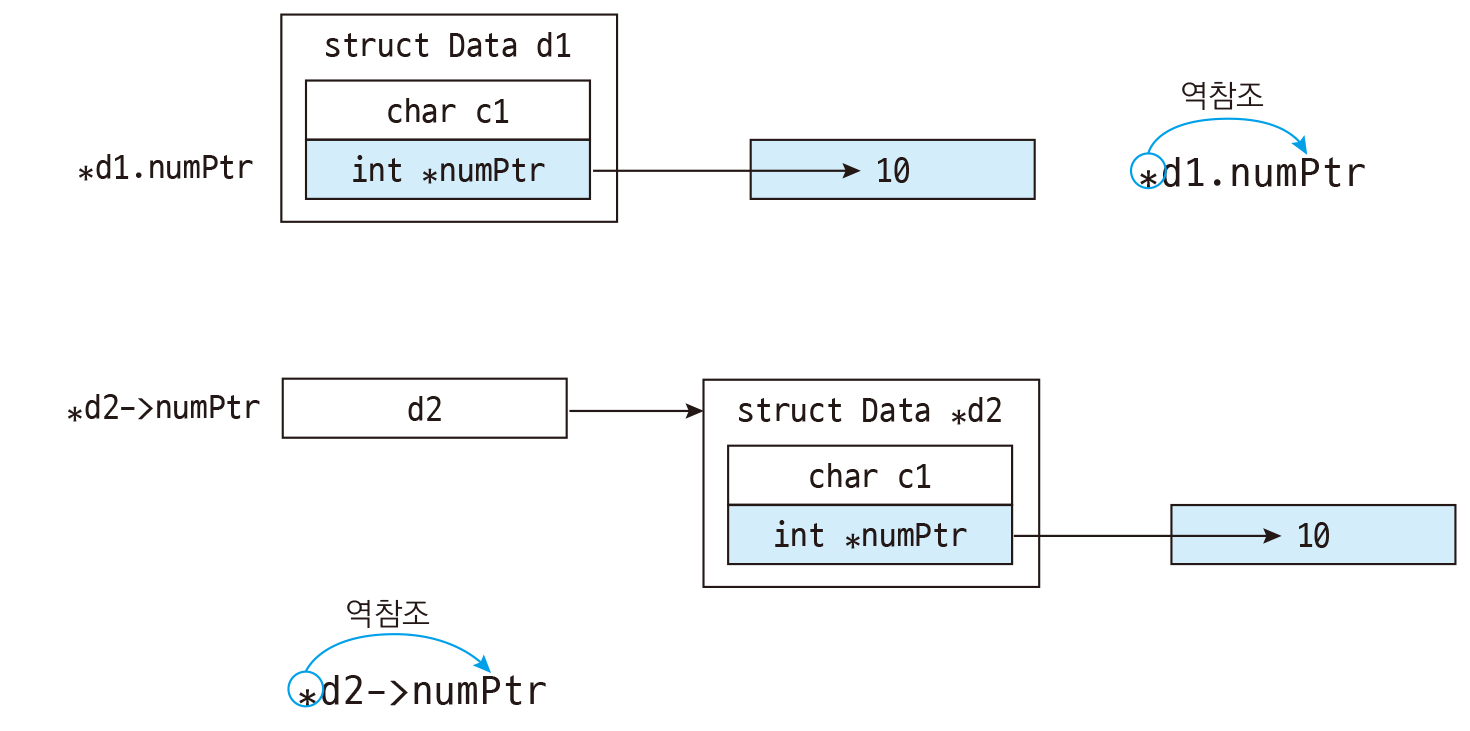
// \*d2->numPtr과 같음

free(d2);

return 0;

}

구조체 변수 d1의 멤버 numPtr을 역참조 하는 방법과 구조체 포인터 d2의 멤버 numPtr을 역참조 하는 방법을 그림으로 표현하면 다음과 같은 모양이 됩니다.

▼ 그림 49-1 구조체 멤버가 포인터일 때 역참조하기

만약 역참조한 것을 괄호로 묶으면 어떻게 될까요? 이렇게 하면 구조체 변수를 역참조한 뒤 멤버에 접근한다는 뜻이 됩니다. \*(\*d2).numPtr처럼 구조체 포인터를 역참조하여 numPtr에 접근한 뒤 다시 역참조할 수도 있습니다.

* **(\*구조체포인터).멤버**
* **\*(\*구조체포인터).멤버**

d2->c1 = 'a';

printf("%c\n", (\*d2).c1); // a: 구조체 포인터를 역참조하여 c1에 접근

// d2->c1과 같음

printf("%d\n", \*(\*d2).numPtr); // 10: 구조체 포인터를 역참조하여 numPtr에 접근한 뒤 다시 역참조

// \*d2->numPtr과 같음

여기서 (\*d2).c1은 d2->c1과 같고, \*(\*d2).numPtr은 \*d2->numPtr과 같습니다. 즉, 구조체 포인터를 역참조한 뒤 괄호로 묶으면 -> 연산자에서 . 연산자를 사용하게 되므로 포인터가 일반 변수로 바뀐다는 뜻입니다. 역참조의 원리와 같죠.

▼ 그림 49-2 구조체 포인터를 역참조한 뒤 괄호로 묶기