

# 포인터(Pointer)



# Visual Studio 2022



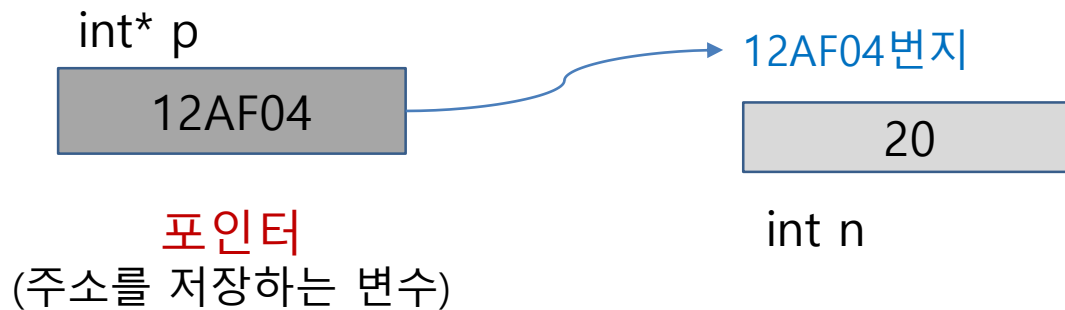
# 포인터(Pointer)

## ➤ 포인터란?

모든 메모리는 주소(address)를 갖는다. 이러한 **메모리 주소를 저장**하기 위해 사용되는 변수를 포인터 변수라 한다.

포인터 변수를 선언할 때에는 데이터 유형과 함께 '\*' 기호를 써서 나타낸다.

(예) 택배 주소만 있으면 집을 찾을 수 있다.



# 포인터(Pointer)

## ➤ 포인터 변수의 선언 및 값 저장

### ▪ 선언

**자료형\*** 포인터 이름

```
char* c;    // char형 포인터  
int* n;     // int형 포인터  
double* d;  // double형 포인터
```

- 포인터의 크기 – 모든 자료형에서 8바이트로 동일하다.  
포인터에 저장할 수 있는 값은 메모리 번지뿐이며, 따라서 모든 포인터 변수는 동일한 크기의 메모리가 필요함.

sizeof(포인터)



# 포인터(Pointer)

- 정수형 포인터 변수

```
int n;  
int* pn;  
  
n = 3;  
pn = &n;  
  
printf("변수의 값: %d\n", n);  
printf("변수의 메모리 번지: %x\n", &n);  
printf("포인터 변수의 값: %x, \n", pn);  
printf("포인터의 메모리 번지: %x\n", &pn);  
printf("포인터가 가리키는 메모리의 값: %d\n", *pn); //역참조  
  
//자료형의 크기  
printf("변수의 자료형의 크기: %dByte\n", sizeof(n));  
printf("포인터의 자료형의 크기: %dByte\n", sizeof(pn));
```

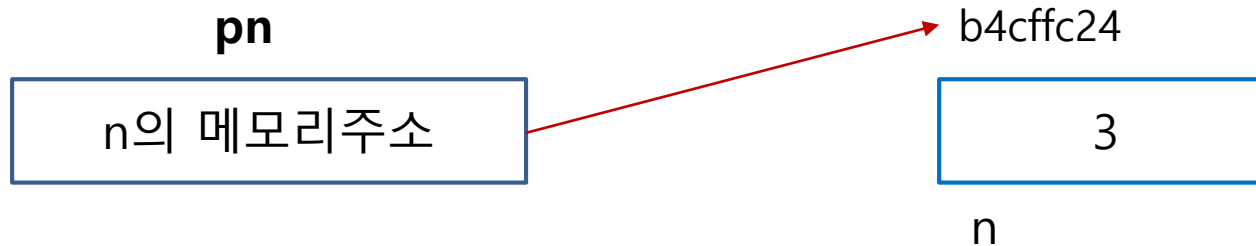


# 포인터(Pointer)

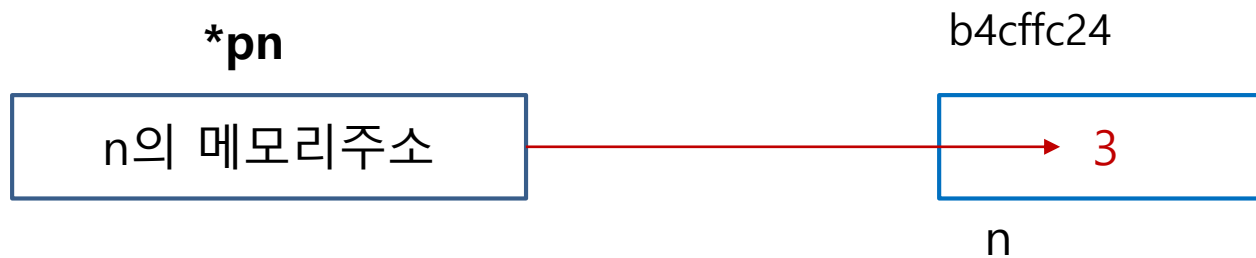
- 역참조 연산자( \* )

포인터를 선언할 때도 \*를 사용하고, 역참조 할때도 \*를 사용

- 포인터는 변수의 주소만 가리킴



- 역참조는 주소에 접근하여 값을 가져옴



# 포인터(Pointer)

- 문자형 포인터 변수

```
char c;  
char* pc;  
  
c = 'A';  
pc = &c;  
  
printf("변수의 값: %c\n", c);  
printf("변수의 메모리 번지: %x\n", &c);  
printf("포인터의 값: %x\n", pc);  
printf("포인터의 메모리 번지: %x\n", &pc);  
printf("포인터가 가리키는 메모리 값: %c\n", *pc);  
  
//변수와 포인터의 자료형의 크기  
printf("변수의 자료형 크기: %dByte\n", sizeof(c));  
printf("포인터의 자료형 크기: %dByte\n", sizeof(pc));
```



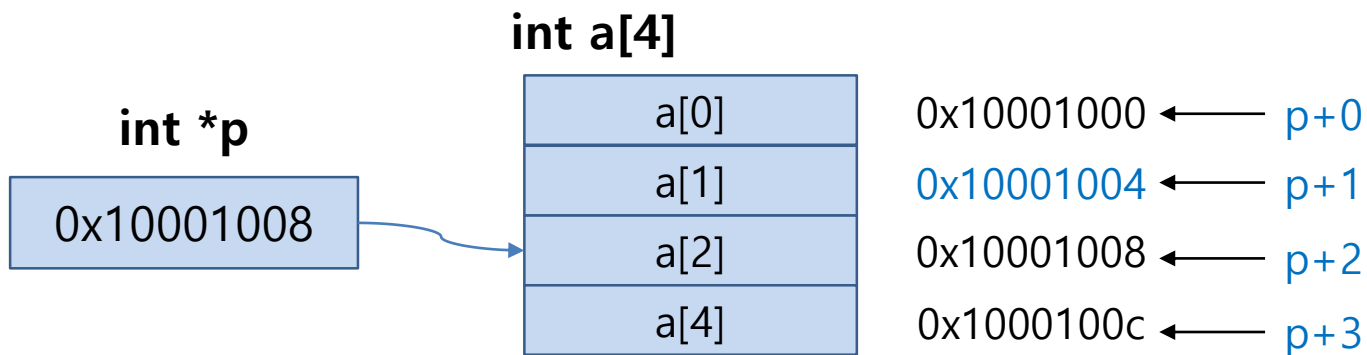
# 배열과 포인터

- 배열과 포인터

- 배열은 데이터를 연속적으로 메모리에 저장한다.

포인터 역시 메모리에 데이터를 저장하거나 저장된 데이터들을 읽어올수 있다.

```
int a[4], *p;    // 배열 a와 포인터 p선언
p = &a[2]        // p에 배열의 두 번째 항목 주소 복사
```



# 배열과 포인터(Pointer)

- 배열과 포인터의 연산(int형 포인터)

```
int a[4] = { 10, 20, 30, 40 };
int* pa;
int i;

printf("a[0]의 값은 %d\n", a[0]);
printf("a[0]의 메모리 번지는 %x\n", &a[0]);
printf("배열의 이름 a는 %x\n", a);

printf("a[1]의 값은 %d\n", a[1]);
printf("a[1]의 메모리 번지는 %x\n", &a[1]);
printf("배열의 이름 a + 1는 %x\n", a + 1);

for (i = 0; i < 4; i++)
{
    printf("%d %x %x\n", a[i], &a[i], a + i);
}
```





# 배열과 포인터(Pointer)

- 배열과 포인터의 연산(int형 포인터)

```
pa = a;

for (i = 0; i < 4; i++)
{
    printf("%d %x %x\n", pa[i], &pa[i], pa + i);
}

printf("=====\n");

printf("포인터 pa의 값은 %x\n", pa);
printf("포인터 *pa이 가리키는 메모리의 값은 %d\n", *pa);

printf("포인터 pa + 1의 값은 %x\n", pa + 1);
printf("포인터 *pa + 1이 가리키는 메모리의 값은 %d\n", *pa + 1);

for (i = 0; i < 4; i++)
{
    printf("%x %d\n", pa + i, *(pa + i));
}
```



# 배열과 포인터(Pointer)

- 배열과 포인터의 연산(int형 포인터)

```
//정수형 포인터 배열 연산
int x = 10, y = 20, z;
int total;

//1. 포인터 배열 선언후 초기화
/*int* arr[3];
arr[0] = &x;
arr[1] = &y;
arr[2] = &z;*/

//2. 포인터 배열 선언과 동시 초기화
int* arr[3] = { &x, &y, &z };

*arr[2] = *arr[0] + *arr[1]; //역참조로 계산

printf("결과값 : %d\n", *arr[2]);
printf("결과값 : %d\n", z);
```



# 배열과 포인터(Pointer)

- 문자열과 포인터

문자열은 문자들이 메모리 공간에 연속적으로 저장되어 있어서 주소로 관리되고, 문자열의 시작 주소를 알면 모든 문자열에 접근할 수 있다.

```
char msg[] = "Good Luck";
char* p = msg;
int i;

printf("%s\n", msg);

printf("문자열의 크기: %dByte\n", sizeof(msg));
printf("문자열 포인터의 크기: %dByte\n", sizeof(p));

printf("문자열 배열이 가리키는 메모리 번지: %x\n", msg);
printf("문자열 포인터가 가리키는 메모리 번지: %x\n", p);
```



# 배열과 포인터(Pointer)

- 문자열과 포인터

```
printf("%s\n", p);
printf("%s\n", p + 1); //다음 문자 출력
printf("%s\n", p + 2);
printf("%s\n", p + 3);

printf("=====\n");

//문자열의 크기
printf("%d %d\n", sizeof(msg), sizeof(msg[0]));
int size = sizeof(msg) / sizeof(msg[0]);
printf("%d\n", size);

//포인터 역참조로 출력
for (i = 0; i < size; i++)
{
    printf("%c", *(p + i));
}
```



# 포인터(Pointer)를 사용한 문자열 처리

```
char a[20];  
char* b;  
  
printf("문자열을 입력하세요: ");  
//scanf("%s", a);  
scanf_s("%s", a, sizeof(a));  
  
b = a;  
printf("저장된 문자열: %s\n", b);
```

```
//문자열 포인터 선언  
char* id = "CLOUD";  
  
printf("%s\n", id);  
printf("%s\n", id + 1);  
printf("%s\n", id + 2);  
printf("%s\n", id + 3);  
printf("%s\n", id + 4);
```



# 포인터(Pointer)

- 함수의 매개변수로 포인터 사용하기

```
void changeArray(int* ptr)
{
    ptr[1] = 50;
}

int main()
{
    //배열 요소 변경 - 포인터 사용
    int arr[] = { 10, 20, 30 };

    changeArray(arr); //int *ptr = arr

    for (int i = 0; i < 3; i++)
    {
        printf("%d\n", arr[i]);
    }

    return 0;
}
```



# 함수에서 포인터의 전달

- Call-by-value(값에 의한 호출) vs Call-by-reference(참조에 의한 호출)

값을 매개체로 함수호출

CallByVal(int i)



CallByVal(n)

주소를 매개체로 함수 호출

CallByRef(int \*i)



CallbyRef(&n)



# 값 & 참조에 의한 호출

- Call-By-Value(값에 의한 호출)

```
void CallByValue(int n)
{
    printf("함수 내에서 값 변경전: %d\n", n);
    n++;
    printf("함수 내에서 값 변경후: %d\n", n);
}
```

```
int main()
{
    int num = 10;

    printf("main 함수 내에서 함수 호출전: %d\n", num);
    CallByValue(num);

    printf("main 함수 내에서 함수 호출후: %d\n", num);
    return 0;
}
```

```
main 함수 내에서 함수 호출전: 10
함수 내에서 값 변경전: 10
함수 내에서 값 변경후: 11
main 함수 내에서 함수 호출후: 10
```





# 값 & 참조에 의한 호출

- Call-By-Reference(참조에 의한 호출)

```
void CallByReference(int *p)
{
    printf("함수 내에서 값 변경전: %d\n", *p);
    (*p)++;
    /* *p = *p + 1;
    printf("함수 내에서 값 변경후: %d\n", *p);
}

int main()
{
    int num = 10;
    int* pn = &num;

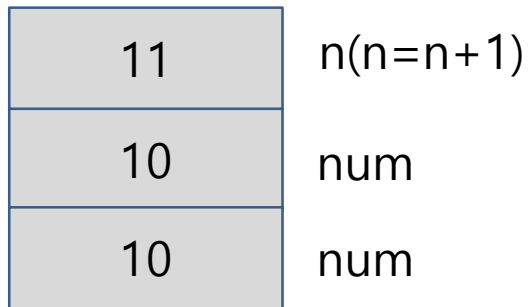
    printf("main 함수 내에서 함수 호출전: %d\n", num);
    CallByReference(pn);
    printf("main 함수 내에서 함수 호출후: %d\n", num);
    return 0;
}
```

```
main 함수 내에서 함수 호출전 : 10
함수 내에서 값 변경전 : 10
함수 내에서 값 변경후 : 11
main 함수 내에서 함수 호출후 : 11
```

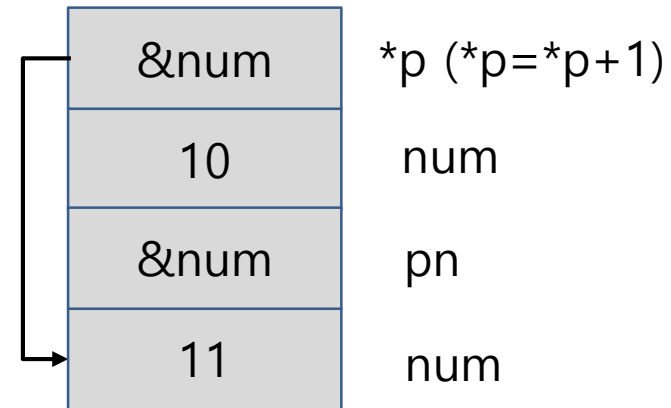


# 값 & 참조에 의한 호출

- Call-By-Value(값에 의한 호출) vs Call-By-Reference(참조에 의한 호출)



매개변수  $n$ 은 호출된 후 11을 반환하고 소멸됨,  
main()의 지역변수 num은 그대로 10을 유지함



매개 포인터  $p$ 는 주소에 저장된 값에 접근하고 역참조 계산으로 num은 11이 됨.  
호출된 후  $p$ 의 메모리 공간은 소멸됨.



# 실습 문제

- 포인터 배열에서 최대값 찾기

```
int findMax(int[], int);  
int findMax(int*, int);  
int main() {  
  
    //최대값 찾기  
    int arr[] = { 21, 35, 71, 2, 97, 66 };  
  
    int max = findMax(arr, 6);  
  
    printf("최대값: %d\n", max);  
  
    return 0;  
}
```



# 실습 문제

- 포인터 배열에서 최대값 찾기

```
//매개변수를 배열로 전달
int findMax(int arr[], int len)
{
    int maxVal = arr[0];

    //printf("%d\n", maxVal);

    for (int i = 1; i < len; i++)
    {
        if (arr[i] > maxVal)
        {
            maxVal = arr[i];
        }
    }

    return maxVal;
}
```

```
//매개변수를 포인터로 전달
int findMax(int* arr, int len)
{
    int maxVal = *(arr + 1);

    //printf("%d\n", maxVal);

    for (int i = 1; i < len; i++)
    {
        if (*(arr + i) > maxVal)
        {
            maxVal = *(arr + i);
        }
    }

    return maxVal;
}
```



# 실습 문제

- 포인터 배열에서 최대값의 위치 찾기

```
int findMaxIdx(int[], int);
int findMaxIdx(int*, int);
int main() {

    //최대값 찾기
    int arr[] = { 21, 35, 71, 2, 97, 66 };

    int maxIdx = findMaxIdx(arr, 6);

    printf("최대값의 위치(인덱스): %d\n", maxIdx);

    return 0;
}
```



# 실습 문제

- 포인터 배열에서 최대값의 위치 찾기

```
//매개변수를 배열로 전달
int findMaxIdx(int arr[], int len)
{
    int maxIdx = 0;

    for (int i = 1; i < len; i++)
    {
        if (arr[i] > arr[maxIdx])
        {
            maxIdx = i;
        }
    }

    return maxIdx;
}
```

```
//매개변수를 포인터로 전달
int findMaxIdx(int* arr, int len)
{
    int maxIdx = 0;

    for (int i = 1; i < len; i++)
    {
        if (*(arr + i) > *(arr + maxIdx))
        {
            maxIdx = i;
        }
    }

    return maxIdx;
}
```



## ◆ 동적 할당의 필요성

배열의 크기는 프로그램이 컴파일되는 과정에서 결정된다. 이를 "정적할당"이라 한다. 정적할당을 사용하면 간혹 메모리가 낭비되는 경우가 발생한다. 만약 프로그램이 실행되는 동안 필요한 크기만큼의 배열을 생성할 수 있다면 메모리의 낭비를 줄일 수 있다.

- 정수형 값 10개를 저장하기 위한 1차원 배열 생성(동적 할당)

```
int* array = (int *)malloc(sizeof(int) * 10);
```

동적으로 할당되는 배열의 메모리 주소는 프로그램이 실행되기 이전에는 알 수 없는 값이므로 이 주소를 저장하기 위해 포인터가 필요하다.

**malloc(memory allocation)** 함수는 필요한 메모리의 크기를 바이트 단위로 할당할 수 있도록 해준다. malloc 함수는 할당된 메모리 주소를 반환한다. 반환되는 메모리 번지에 어떤 유형의 데이터를 저장할 것인지 지정해 주어야한다.

**free** 함수는 동적으로 할당된 메모리를 시스템에 반납하도록 해 준다.



# 동적 메모리 할당

## ◆ 동적 할당 사용

```
// 정수형 배열 10개 선언 - 정적 할당
//int pn[10];

// 정수형 배열 10개 선언 - 동적 할당
int* pn = (int *)malloc(sizeof(int) * 10);
int i;

for (i = 0; i < 10; i++)
{
    pn[i] = i * 2;
}

for (i = 0; i < 10; i++)
{
    printf("%d\n", pn[i]);
}

free(pn); //할당된 메모리 해제(반납)
```





# 동적 메모리 할당

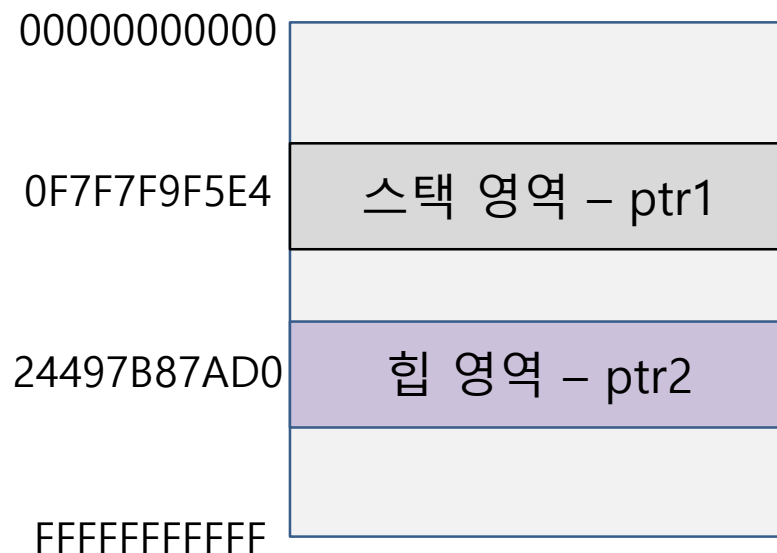
```
int num1 = 10;
int* ptr1; //스택 영역에 위치함
int* ptr2; //힙 영역에 위치함

ptr1 = &num1;

//정수형 배열 3바이트 선언
ptr2 = (int*)malloc(sizeof(int) * 3);
if (ptr2 == NULL)
{
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1); //강제 종료
}

printf("%p\n", ptr1);
printf("%p\n", ptr2);

free(ptr2);
```



# 동적 메모리 할당 예제

## ◆ 정수형 배열 4개 동적 할당

```
int* ip;
int i;

ip = (int *)malloc(sizeof(int) * 4);

if (ip == NULL)
{
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1);
}
//배열로 저장하기
ip[0] = 10;
ip[1] = 20;
ip[2] = 30;
ip[3] = 40;

for (i = 0; i < 4; i++)
{
    printf("%d\n", ip[i]);
}
```



# 동적 메모리 할당 예제

## ◆ 정수형 배열 4개 동적 할당

```
//역참조로 저장하기
*ip = 50;
*(ip + 1) = 60;
*(ip + 2) = 70;
*(ip + 3) = 80;

for (i = 0; i < 4; i++)
{
    printf("%d %d\n", ip[i], *(ip + i));
}

free(ip);
```



# 동적 메모리 할당 예제

## ◆ 문자형 배열 30개 동적 할당

```
char* pc;  
int i;  
pc = (char *)malloc(sizeof(char) * 30);  
if (pc == NULL)  
{  
    printf("동적 메모리 할당에 실패했습니다.\n");  
    exit(1);  
}  
  
for (i = 0; i < 26; i++)  
{  
    *(pc + i) = 'a' + i;  
}  
*(pc + i) = '\0'; //맨 뒤에 NULL 추가  
  
printf("%s", pc);  
  
free(pc);
```



## 8장. 구조체

# Visualstudio 2019



# 구조체의 개념

## ◆ 구조체는 왜 필요할까?

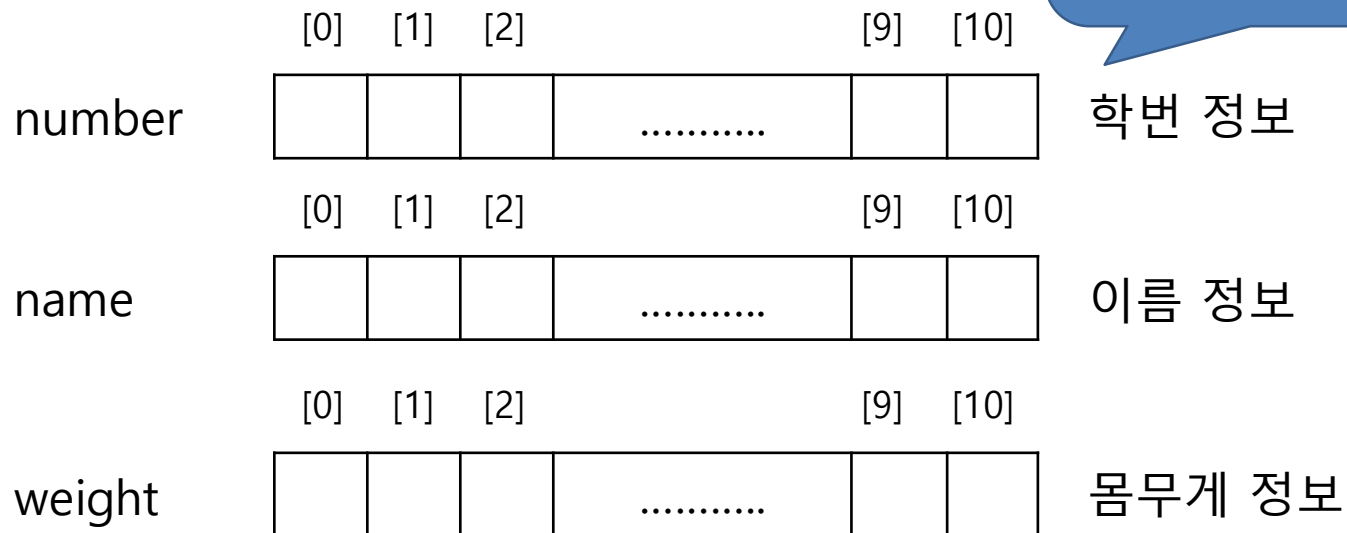
학생 10명의 학번과 이름, 몸무게 정보 저장 – 배열 자료형 이용

```
int number[10];
```

```
char name[20];
```

```
double weight[10];
```

정보가 흩어져서 저장되는 한계 발생



# 구조체란 무엇인가?

## ◆ 구조체(structure)란?

다양한 자료형을 그룹화하여 하나의 변수로 처리할 수 있게 만든 자료형이다. 개발자가 다양한 정보를 저장하기 위해 필요에 따라 생성하는 자료형을 사용자 정의 자료형 또는 구조체라 한다.

- 구조체 정의

```
struct 구조체이름{  
    자료형 멤버이름;  
};
```

- 객체 생성

```
struct 구조체이름 변수이름;
```



# 구조체의 정의 및 사용

- ◆ 구조체 정의 - 멤버 변수는 일반 변수처럼 초기화 할 수 없음

```
struct Person {  
    //이름, 나이, 키  
    char name[20];  
    int age;  
    float height;  
};
```

int n → 정수  
4byte

struct Person;

문자	정수	실수
20byte	4byte	4byte

- ◆ 구조체 객체(변수) 생성

```
struct Person p1;
```





# 구조체의 정의 및 사용

## ◆ 구조체 객체 생성 및 사용

```
struct Person p1; //구조체 객체 선언

//p1.name = "알파고"; //컴파일 오류
//strcpy(변수, 문자열) - 문자열을 복사
strcpy(p1.name, "알파고");
p1.age = 11;
p1.height = 171.9f;

printf("이름: %s\n", p1.name);
printf("나이: %d\n", p1.age);
printf("키: %.1f\n", p1.height);
```



# 구조체 배열

## ◆ 구조체 배열 – 객체를 여러 개 생성

```
//구조체 배열 선언
struct Person p[3] = {
    {"이산", 15, 171.9f},
    {"한강", 35, 163.3f},
    {"박봄", 22, 178.4f},
};
int i;

//p[0]의 정보
/*printf("이름: %s\n", p[0].name);
printf("나이: %d\n", p[0].age);
printf("키: %.1f\n", p[0].height);*/

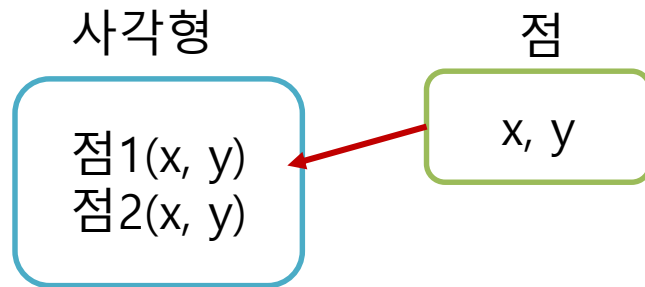
for (i = 0; i < 3; i++)
{
    printf("이름: %s, 나이: %d, 키: %.1f\n",
        p[i].name, p[i].age, p[i].height);
}
```



# 구조체 참조 관계

## ◆ 구조체 참조 관계

구조체의 멤버 변수가 다른 구조체의 객체인 관계



```
struct Point
{
    //점의 좌표
    int x;
    int y;
};

struct Rectangle
{
    //구조체 Point의 객체 생성
    struct Point p1;
    struct Point p2;
};
```



# 구조체 참조 관계

## ◆ 구조체 참조 관계

```
//사각형 객체 선언
struct Rectangle rect;

rect.p1.x = 1;
rect.p1.y = 5;

rect.p2.x = 5;
rect.p2.y = 1;

printf("점1(%d, %d), 점2(%d, %d)\n",
      rect.p1.x, rect.p1.y, rect.p2.x, rect.p2.y);
```



# 구조체 typedef 키워드 사용

- **typedef struct** 구조체

```
typedef struct {  
    자료형 멤버이름;  
} 구조체이름;
```



```
typedef struct {  
    char name[20],  
    int age;  
    float height;  
} Person;
```

```
Person p1;
```



# 구조체 typedef 키워드 사용

- 구조체 변수를 선언할때 struct 키워드 생략

```
typedef struct
{
    //이름, 나이, 키
    char name[20];
    int age;
    float height;
}Person;
```

```
Person p1; //구조체 객체 선언

strcpy(p1.name, "알파고");
p1.age = 11;
p1.height = 171.9f;

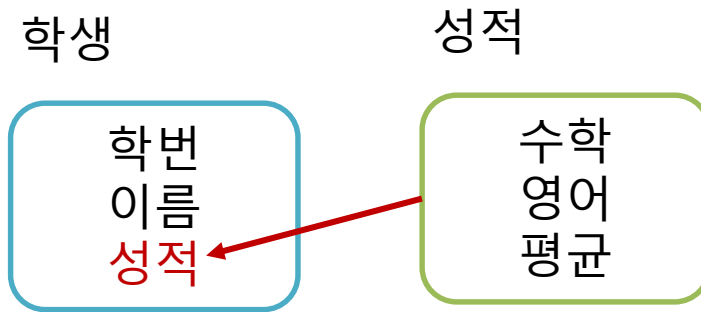
printf("이름: %s\n", p1.name);
printf("나이: %d\n", p1.age);
printf("키: %.1f\n", p1.height);
```



# 구조체 참조 관계

## ◆ 구조체 참조 관계

- 학생이 성적을 참조하는 관계.



Student.h

```
// 성적 구조체
typedef struct
{
    int math;    //수학
    int eng;     //영어
    double avg;  //평균
}Score;

//학생 구조체
typedef struct
{
    int number; //학번
    char name[20]; //이름
    Score score; //점수
}Student;
```



# 구조체 참조 관계

## ◆ 구조체 참조 관계

```
/*Student s1;
...
s1.number = 101;
strcpy(s1.name, "알파고");
s1.score.math = 99;
s1.score.eng = 90;*/

//객체 생성(초기화)
Student s1 = { 101, "알파고", {99, 90, 0.0} };

//성적의 평균
s1.score.avg = (double)(s1.score.math + s1.score.eng) / 2;

//학생의 정보
printf("학번: %d, 이름: %s\n", s1.number, s1.name);
printf("수학: %d, 영어: %d\n", s1.score.math, s1.score.eng);

printf("평균: %.1f\n", s1.score.avg);
```

ScoreMain.c





# 구조체 배열 사용

- 성적 관리 프로그램

1. 성적 구조체 정의(멤버변수: 수학점수, 영어점수)
2. 학생 구조체 정의(학번, 이름, 성적)
3. 학생 3명의 구조체 배열 생성 후 학번, 이름, 수학 점수, 영어 점수 입력
4. 학생 정보 출력은 함수를 정의하고 호출
5. 수학과 영어의 평균 계산하고 출력

```
===== 성적 관리 프로그램 =====
```

```
학 번 입력 : 1
```

```
1번째 학생의 이름 입력 : 이산
```

```
수학점수 입력 : 95
```

```
영어점수 입력 : 86
```

```
학 번 입력 : 2
```

```
2번째 학생의 이름 입력 : 한강
```

```
수학점수 입력 : 90
```

```
영어점수 입력 : 99
```

```
학 번 입력 : 3
```

```
3번째 학생의 이름 입력 : 강하늘
```

```
수학점수 입력 : 88
```

```
영어점수 입력 : 77
```

```
=====
```

```
학 번      이름      수학      영어
```

```
1          이산      95       86
```

```
2          한강      90       99
```

```
3          강하늘    88       77
```

```
=====
```

```
수학 평균 : 91.0, 영어 평균 : 87.3
```



# 구조체 배열 사용

- 성적 관리 프로그램

```
//성적 구조체
typedef struct
{
    int math;
    int eng;
}Score;

//학생 구조체
typedef struct
{
    int number;    //학번
    char name[20]; //이름
    Score score;   //성적
}Student;
```



# 구조체 배열 사용

- 성적 관리 프로그램

```
void showStudentInfo(Student student);
int main()
{
    Student s[3];
    int i;
    int total[2] = { 0, 0 };
    double avg[2] = { 0.0, 0.0 };

    printf("===== 성적 관리 프로그램 ===== \n");
    //입력
    for (i = 0; i < 3; i++)
    {
        printf("학번 입력: ");
        scanf("%d", &s[i].number);

        printf("%d번째 학생의 이름 입력: ", i+1);
        scanf("%s", s[i].name);

        printf("수학점수 입력: ");
        scanf("%d", &s[i].score.math);

        printf("영어점수 입력: ");
        scanf("%d", &s[i].score.eng);
    }
```



# 구조체 배열 사용

- 성적 관리 프로그램

```
printf("=====\n");  
//출력  
printf("학번\t이름\t 수학\t영어\n");  
for (i = 0; i < 3; i++)  
{  
    total[0] += s[i].score.math;    //수학 합계  
    total[1] += s[i].score.eng;    //영어 합계  
  
    showStudentInfo(s[i]); //학생 정보 호출  
}  
//평균 계산  
avg[0] = (double)total[0] / 3;    //수학 평균  
avg[1] = (double)total[1] / 3;    //국어 평균  
printf("=====\n");  
  
printf("수학 평균: %.1f, 영어 평균: %.1f\n", avg[0], avg[1]);  
  
return 0;  
}
```



# 구조체 배열 사용

- 성적 관리 프로그램

함수의 파라미터로  
구조체 배열 사용

```
void showStudentInfo(Student student)
{
    printf("%d\t%s\t%d\t%d\n",
           student.number, student.name, student.score.math, student.score.eng);
}
```



# 구조체 포인터 변수

- 구조체 포인터 사용

```
Person p1 = { "알파고", 11, 171.9f };  
Person* ptr = &p1; //구조체 포인터 선언
```

구조체 변수가 닷(.) 연산자를 사용하는 반면, 구조체의 포인터 변수는 참조 연산자(->)를 사용한다.

```
ptr->name
```

- 구조체 포인터 사용 필요성

1. 구조체 포인터 변수 크기가 작아서 효율적이다.
2. 동적 메모리 할당이 가능하다.

```
Person* p = (Person*)malloc(sizeof(Person) * 3);
```



# 구조체 포인터 변수

- 구조체 포인터 사용

```
typedef struct
{
    char name[20];
    int age;
    float height;
}Person;

int main()
{
    Person p1 = { "알파고", 11, 171.9f };
    //구조체 포인터 선언
    Person* ptr = &p1;

    printf("이름: %s\n", ptr->name);
    printf("나이: %d\n", ptr->age);
    printf("키: %.1f\n", ptr->height);

    return 0;
}
```



# 구조체 포인터 변수

- 구조체 배열 동적 할당

```
//구조체 배열 동적 할당
Person* p = (Person*)malloc(sizeof(Person) * 3);
if (p == NULL)
{
    printf("동적 메모리 할당에 실패했습니다.\n");
    exit(1);
}
//Person 1명 생성
strcpy(p->name, "이산");
p->age = 20;
p->height = 171.3f;

//Person 2명 생성
strcpy((p + 1)->name, "한강");
(p + 1)->age = 35;
(p + 1)->height = 163.5f;

//Person 3명 생성
strcpy((p + 2)->name, "박봄");
(p + 2)->age = 29;
(p + 2)->height = 173.5f;
```





# 구조체 포인터 변수

- 구조체 배열 동적 할당

```
//사람의 정보 출력
/*printf("이름: %s, 나이: %d, 키: %.1f\n",
        p->name, p->age, p->height);
|
|
printf("이름: %s, 나이: %d, 키: %.1f\n",
|   (p+1)->name, (p+1)->age, (p+1)->height);*/


for (int i = 0; i < 3; i++)
{
    printf("이름: %s, 나이: %d, 키: %.1f\n",
|       (p + i)->name, (p + i)->age, (p + i)->height);
}

free(p); //구조체 포인터 해제
```



# 실습 문제

=== Book 구조체를 만들어서 책 2권을 출력하는 프로그램을 작성하세요 ===

실행결과 

```
번호 : 201, 제목 : 모두의 C언어  
번호 : 202, 제목 : 채식주의자
```



# 실습 문제

- 구조체 배열과 포인터 변수

```
//과일 구조체
typedef struct
{
    char name[20];
    int quantity;
    char* type;
}Fruit;
```

```
//2차원 문자열 배열
//char types[][10] = { "Apple", "Banana", "Orange" };

//포인터 배열
char* types[] = { "Apple", "Banana", "Orange" };

//Fruit 객체 생성
Fruit f = { "Daegu Apple", 10, types[0] };

printf("Fruit Name: %s\n", f.name);
printf("Quantity: %d\n", f.quantity);
f.type = "Kiwi";
printf("Fruit Type: %s\n", f.type);
```



# 실습 문제

- 구조체 배열과 포인터 변수

```
//포인터 객체 생성
Fruit* ptr = &f; //f(객체)를 ptr(포인터)에 대입

printf("Fruit Name: %s\n", ptr->name);
printf("Quantity: %d\n", ptr->quantity);

ptr->type = "Kiwi"; //type 변경(업데이트)
printf("Fruit Type: %s\n", ptr->type);
```

