인트아이 정연한

2023-1 인트아이 C++ 수업 자료

C++ 기초 4강 배열, 포인터

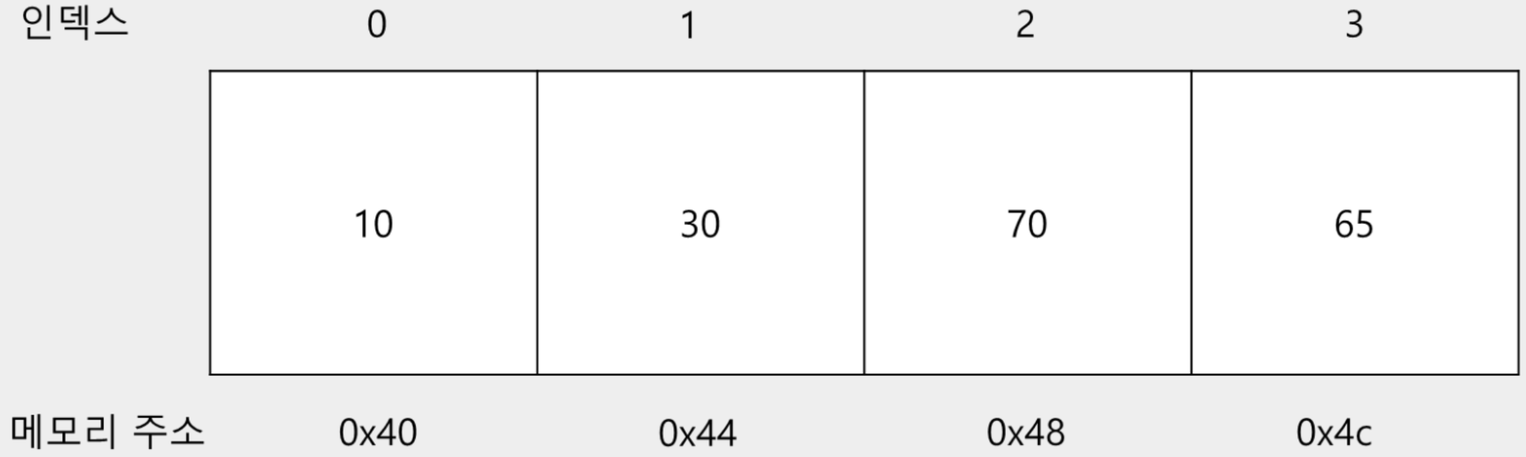
CONTENTS

어디서 많이 들어보긴 했는데...

이전 강의들에서 배열은 간략하게나마 설명되고 사용되었다.
배열이란 간단히 말해서 **데이터를 저장할 공간에 번호를 매겨, 여러 데이터를 저장할 수 있게 한 메모리 공간**을 의미한다.
많은 프로그래밍 언어에서 기본적으로 지원하는 가장 기초적인 자료구조 중 하나로,
C/C++에서는 **대괄호**를 사용해 **배열을 선언**할 수 있다. 이때 사용되는 대괄호는 **배열 선언자** 라고 부른다.

배열은 인덱스를 가지는 연속적인 메모리 공간
따라서 배열은 **선언 시점에 정한 메모리 공간**만을 사용한다.

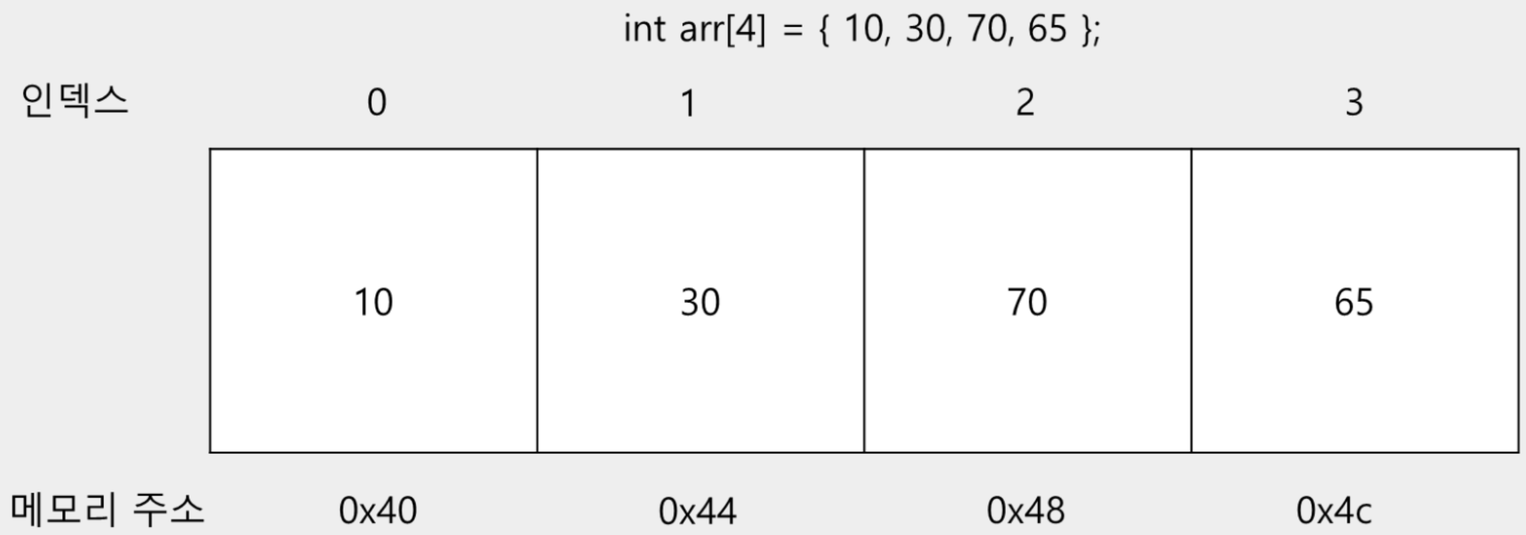
```
int arr[4] = { 10, 30, 70, 65 };
```



배열의 구조

아래 그림을 보면, int array[4] 를 통해 **크기가 4**인 배열을 선언하고,
= { 10, 30, 70, 65 }; 를 통해 배열에 값을 넣어주고 있다.
이때 중괄호 부분을 **이니셜라이저 리스트**(또는 초기화자 리스트) 라고 부르며,
배열과 같은 **자료구조를 초기화** 하는데 사용된다.

배열은 인덱스를 가지는 연속적인 메모리 공간
따라서 배열은 **선언 시점에 정한 메모리 공간**만을 사용한다.

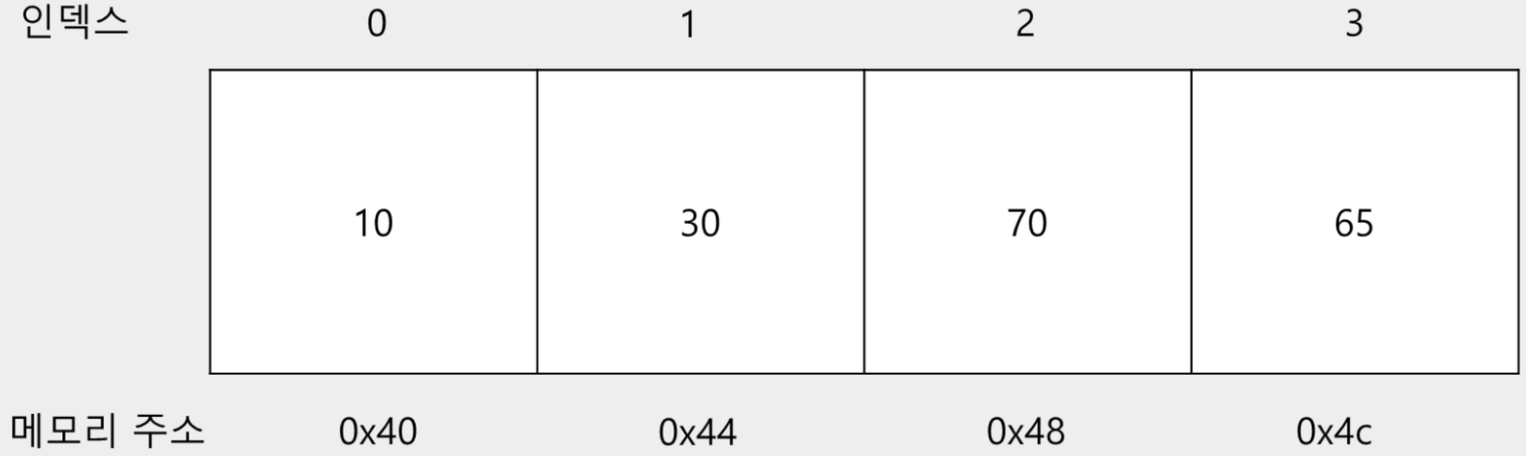


배열의 구조

이렇게 선언 된 배열은 아래 그림과 같이 **메모리 상에 구현**이 된다.
임의의 **메모리 주소**를 **할당**받아 해당 위치부터 배열의 크기만큼 **연속적**으로 **공간을 차지**하게 되는데,
아래 예시에서는 0x40이라는 임의의 위치에 배열이 선언이 되었고, 해당 위치가 **0번 인덱스**,
바로 다음 칸이 **1번 인덱스**, 이런식으로 **가장 끝이 3번 인덱스**가 된다.
int는 크기가 4이므로, **메모리 주소가 4씩 증가**하는 것 역시 볼 수 있다.

배열은 인덱스를 가지는 연속적인 메모리 공간
따라서 배열은 **선언 시점에 정한 메모리 공간**만을 사용한다.

```
int arr[4] = { 10, 30, 70, 65 };
```



코드를 통해 알아보자

배열의 구조에 대해 알아보았으니 이번에는 실제 코드에서 배열이 어떻게 사용되는지에 대해 알아보자.
우선 **배열**은 주로 **반복문과 함께 사용** 된다.
예를 들어 학생들의 학번을 차례대로 입력받고 그것을 정렬해서 다시 출력해야 한다면,
오른쪽과 같이 작성해주면 된다.

```cpp
int main() {
    // 학번을 받을 배열
    int SN[3] = { };

    // 학번 순차적으로 입력
    for (int i = 0; i < sizeof(SN) / sizeof(int); ++i)
        cin >> SN[i];

    // 배열의 이름은 배열의 첫번째 요소의 주소값과 같다.
    // 즉, SN == &SN[0]
    // 따라서 아래는 &SN[0], &SN[3]과 같다.
    sort(SN, SN + 3);

    cout << endl << "정렬 결과: ";
    for (int i = 0; i < sizeof(SN) / sizeof(int); ++i)
        cout << SN[i] << ' ';

    return 0;
}
```

배열은 특히나 **for문**과 궁합이 좋은데, **순차적으로 저장**이 되는 **배열**의 특성과,
순차적으로 증감되는 **for문**의 특성이 대부분의 경우 서로 잘 맞기 때문이다.

for문을 잘 보면 sizeof가 들어가 있는 것을 볼 수 있는데,
sizeof는 괄호 안에 들어간 값의 **크기를 반환**하는 연산자로,
SN의 크기는 **int형 데이터가 3개** 들어가 있는 크기 이므로 이것을 **int의 크기로 나눠**주게 되면
3이 나오게 되어, 이 반복문은 **3회 반복**하여 입력을 받게 된다.

그 다음 sort함수에 의해 정렬이 진행되고, 그 결과는 다시 for문을 통해 출력을 할 수 있다.

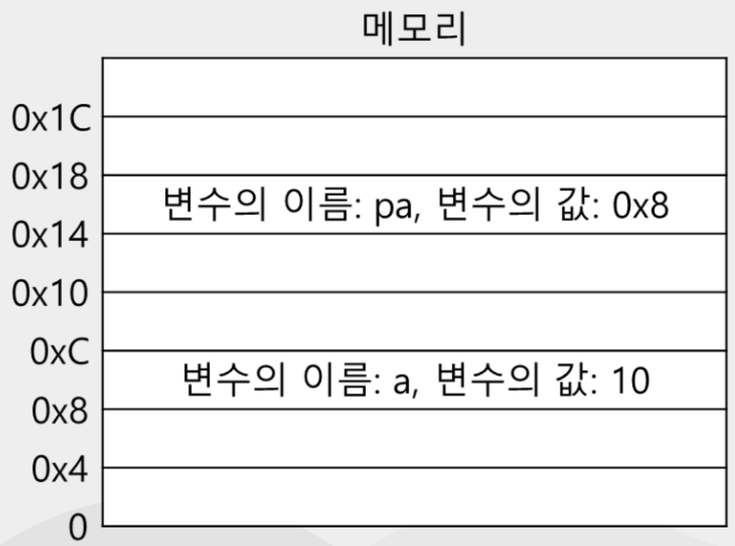
배열은 이것 말고도 다양한 상황에서 쓸 수 있으며, 더 많은 예제는 과제들을 진행하면서 확인해보도록 하자.

```
1220
1217
1222

정렬 결과: 1217 1220 1222
```

포인터란 가리키는 것이다.

포인터는 어렵기로 소문이 났다.
프로그래머의 1차 장벽이라느니, 포인터를 이해 못하면 프로그래밍을 접어야 한다느니, 별별 얘기들이 많은데,
다 필요 없고 간단하게, 포인터는 그냥 **무언가를 가리키는 놈**이다.

자 여기 간단한 메모리가 있다. 이 메모리에 변수를 선언 해줄건데, 우선 int a = 10;을 통해 a라는 변수를 선언해주자.
이 변수는 운영체제가 **메모리 주소 0x8을 할당**을 해줘서 **해당 위치에 변수의 값을 넣어**주었다.
그다음, int* pa = &a;를 통해 pa라는 변수를 선언해주었으며, **메모리 주소 0x14를 할당**받았다.
이 변수는 **포인터 선언자(*)**를 통해 선언되어 **포인터를 담을** 수 있는 **포인터 변수**로 선언되었고,
해당 변수의 값으로는 = &a;를 통해 **a의 포인터**를 받았다.



int a = 10; // 0x8에 할당
int* pa = &a; // 0x14에 할당

포인터란 가리키는 것이다.

자, 고작 포인터 변수 하나 선언했다고 이해가 안되는 말들이 잔뜩 들렸을 것이다.
정리를 좀 해보자. **포인터 변수는 포인터를 담을 수 있는 변수**이다.
그렇다면 **포인터 변수에 담을 포인터**는 어디서 구할까?
정답은 **다른 변수들에 &(앰퍼샌드)를 붙여 구할** 수 있다.
&는 흔히 주소 연산자라고 불린다. 틀린말은 아니다. 실제로 반환 된 값을 봐도 a의 주소값을 잘 반환했으니까,
그럼 왜 &를 통해 포인터를 구한다고 했느냐, 결론부터 말하자면, **주소값의 자료형이 포인터**이기 때문이다.
믿기지 않는다면 오른쪽 코드와 결과를 보자.
int *라고 잘 표시되는 것이 보일 것이다.
따라서 결국 **포인터 변수는 주소값을 받는다**고 해도 맞는말이 되는 것이다.

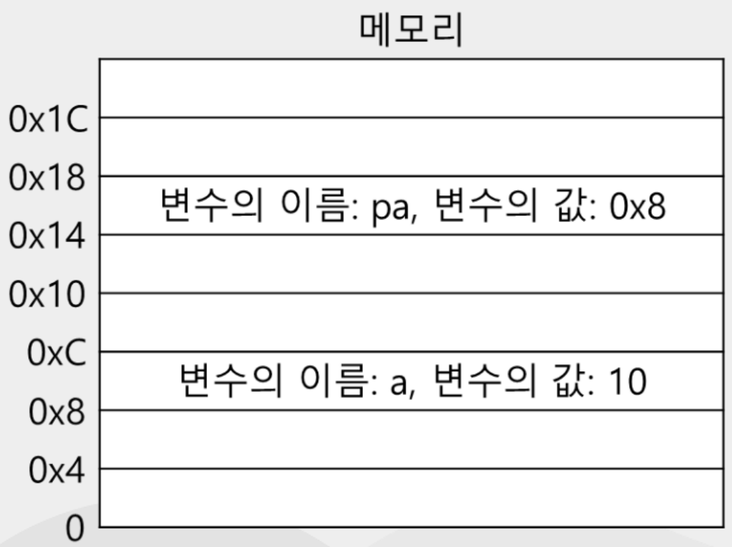
```cpp
int main() {
    int a = 10;
    int* pa = &a;

    cout << typeid(a).name() << endl;
    cout << typeid(&a).name() << endl;
    cout << typeid(pa).name() << endl;
    return 0;
}
```

```
int
int * __ptr64
int * __ptr64
```

64비트 환경에서 실행했을 때

```
int
int *
int *
```

32비트 환경에서 실행했을 때

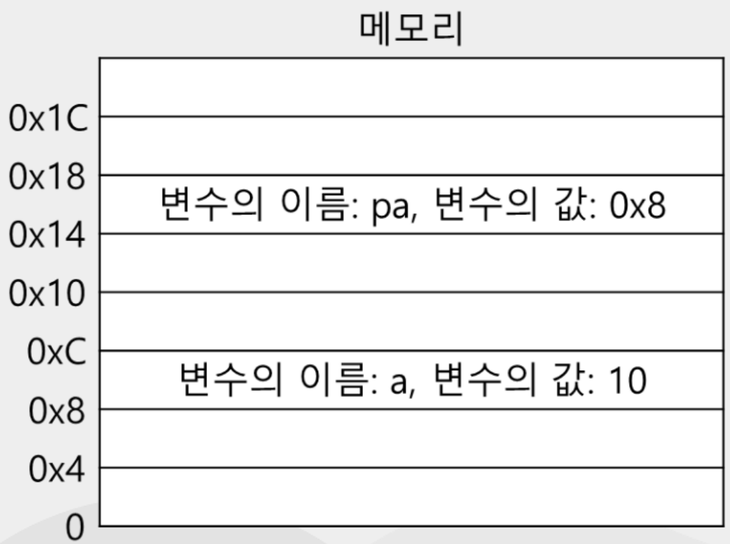


int a = 10; // 0x8에 할당
int* pa = &a; // 0x14에 할당

그런데... 주소값은 크기가 다 동일한데?

주소값은 기본적으로 컴파일 하는 환경에 따라 4바이트(32비트 환경) 또는 8바이트(64비트 환경)이지만,
환경이 같다면 어떤 변수든 주소값은 모두 동일하다.
비유를 하자면, 몸무게가 50kg인 사람과 몸무게가 100kg 사람의 집주소 체계가 서로 다르지 않다는 것과 같다.
다시말해 변수의 크기와 해당 변수의 위치는 전혀 상관관계가 없다는 뜻이다.

그런데 이렇게 되면, 뭔가 이상한 점이 있는데, 그렇다면 굳이 포인터 변수를 선언할때,
앞에다가 자료형을 붙여줄 이유가 없지 않나 하는 생각이 들 수 있다.
포인터 변수를 선언할 때 포인터 선언자 앞에 자료형을 붙여주는 이유는,
해당 포인터가 그 자료형의 주소만 받을 수 있게 한정하는 용도로 쓰인다.
int형 포인터인데 double의 주소값을 받는다든가 이런 경우를 차단하기 위해서 라고 보면 될 것 같다.




```cpp
int main() {
    int a = 10;
    int* pa = &a;

    cout << typeid(a).name() << endl;
    cout << typeid(&a).name() << endl;
    cout << typeid(pa).name() << endl;
    return 0;
}
```

```
int
int * __ptr64
int * __ptr64
```

64비트 환경에서 실행했을 때

```
int
int *
int *
```

32비트 환경에서 실행했을 때

int a = 10; // 0x8에 할당
int* pa = &a; // 0x14에 할당

백준 1546번 평균

문제

세준이는 기말고사를 망쳤다. 세준이는 점수를 조작해서 집에 가져가기로 했다. 일단 세준이는 자기 점수 중에 최댓값을 골랐다. 이 값을 M이라고 한다. 그리고 나서 모든 점수를 점수/M*100으로 고쳤다.
예를 들어, 세준이의 최고점이 70이고, 수학점수가 50이었으면 수학점수는 50/70*100이 되어 71.43점이 된다.
세준이의 성적을 위의 방법대로 새로 계산했을 때, 새로운 평균을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 시험 본 과목의 개수 N이 주어진다. 이 값은 1000보다 작거나 같다. 둘째 줄에 세준이의 현재 성적이 주어진다. 이 값은 100보다 작거나 같은 음이 아닌 정수이고, 적어도 하나의 값은 0보다 크다.

출력

첫째 줄에 새로운 평균을 출력한다. 실제 정답과 출력값의 절대오차 또는 상대오차가 10-2 이하이면 정답이다.

예제 입력 1 복사	예제 출력 1 복사
3 40 80 60	75.0

예제 입력 2 복사	예제 출력 2 복사
3 10 20 30	66.666667

10^{-2} 이하의 오차를 허용한다는 말은 정확히 소수 2번째 자리까지 출력하라는 뜻이 아니다.