

01

배열

02

포인터

03

동적할당



포인터 - 배열



배열이란

- 같은 type의 변수를 연속적으로 저장하기 위해 사용하는 방법
- 선언형식
(type) (변수명)[길이];

ex)

A[6] :

10	20	30	40	50	60
----	----	----	----	----	----

```
int A[6] = {10,20,30,40,50,60};
```



배열이란

- 다양한 배열 선언 형식

```
int arr1[3] = {1, 2, 3};
```

```
int arr2[] = {1, 2, 3};
```

 (길이는 생략 가능)

```
int arr2_c[];
```

 얘는 길이를 몰라서 불가능

```
int arr3[3];
```

 (arr안의 값은 배열이 선언된 자리의 값이 나온다 != 0)

arr3[0] = 1; arr3[1] = 2; arr3[2] = 3; 을 해줘야 arr1과 arr2와 같은값을 갖게된다.

```
int a = 3;
```

```
int arr4[a];
```

 배열의 길이로 변수는 못 들어온다.

변수선언이 전부 끝나고 값을 넣기 때문

실행은 이순서기 때문 int a; -> int arr4[a] -> a = 3

여기는 전부 int형(4바이트) * 3개 = 12바이트를 배열로 선언



배열이란

배열의 초기화

`int arr[3];` 이때 `arr[0], arr[1], arr[2] != 0`

`int arr[3] = {};` 를 통해 0으로 가득 채울 수 있다.

`int arr[3] = {1, 2};` 처럼 크기가 작아도 가능 = `{1,2,0}`

`int arr[3] = {1,2,3,4};` 처럼 크기가 크면 error



배열이란

- 배열에 접근하기(index)

arr = {1, 2, 3}인 arr 2번째 원소에 접근하려면 arr[2-1]을 사용하면 된다.

따라서 arr[1] = 2

arr[3]의 경우에는 arr의 4번째 원소가 존재하지않기때문에 error



이차원배열

- 배열속에 배열이 존재하는것
- 도식화하면 행렬로 볼 수 있다.

• `int arr[2][3] = {{1, 2, 3},
 {4, 5, 6}};`

`int arr[2][3] = {1, 2, 3, 4, 5, 6};` 이렇게 선언한것도 같다.

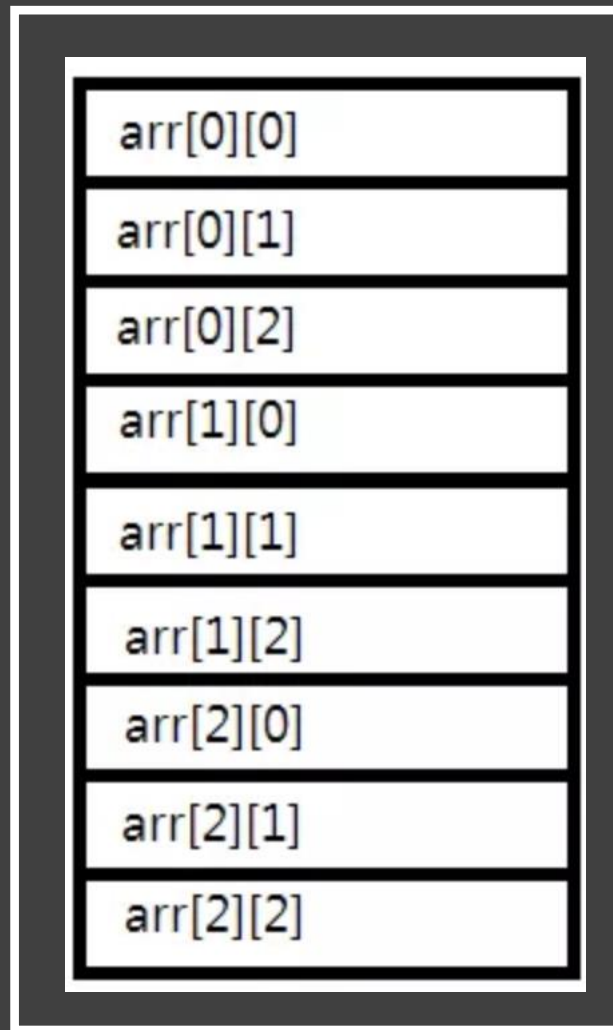
`arr[1][2] = 6, arr[0][1] = 2`
다

각 원소(2개)가 원소3개가지는 배열
이렇게 2차원 배열에서도 값 얻을수 있



이차원배열

- 컴퓨터에 실제로 저장은
이렇게 연속적으로 저장됩니다.



이차원배열

- 다양한 이차원 배열 선언
- `int arr[][3] = {{1, 2, 3}, {4, 5}};` 0해도 `int a[3] = {3}`처럼 0으로 채움
- `int arr[2][] = {{1, 2, 3}, {4, 5, 6}};` 0하는 error 맨 앞 갯수만 생략 가능하다.
- 따라서 `int arr[][2][3] = {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}};` 이런경우는 가능하다.
- `arr = {}`로 0초기화가 가능하며 2중for문을 통해서도 초기화 가능하다



String char [n]

- Char 배열 : ascii값을 저장해놓은 배열
- N개의 방 뒤에는 null(ascii 0)이 존재한다. (총 N+1개의 방을 가진다)
- 따라서 char str[3] = "HYU";

str[0] = "H"=72, str[3] = null = 0

(%c는 앞의 %d는 뒤의값이 나온다. %s는 문자열 전체 출력)

“문자열”, “문자”, ‘문자’(O)

‘문자열’(X)

‘\n’같은 특수기호는 문자취급



String char [n]

- 값 갖고 놀기
- `Char str[] = "abc";` \rightarrow `str = "dbc"`
 `str[0] = 'd' (= 100) (= 'c' + 1)`

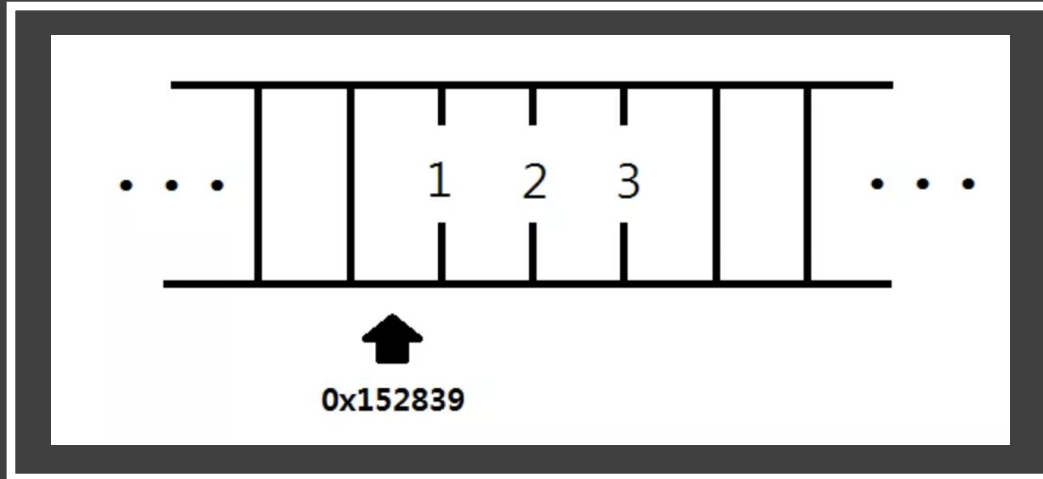


포인터 - 포인터



포인터란

- 모든 데이터는 메모리상 특정한 공간에 저장되어있다.



- 그때 들어간 데이터의 주소를 가리켜서 포인터라고 한다.

포인터란

- 포인터 선언 가르키는 자료형 *변수명
- `int a;`
- `int *p;`



주소 연산자와 주소 역연산자

- 주소 연산자(주소를 출력 &) 따라서 scanf에서는 pointer를 받아온다.
- `char a = 100;`
- `char *p;`
- `p = &a;`
- `&a`는 `a`의 주소인 `p`를 출력, `*p`는 `p`주소에 담긴 값인 `a`를 출력
- 주소 역 연산자(주소에 담긴 값을 출력*)



포인터와 배열, 연산

- `int arr[3] = {1, 2, 3}`
- `int *p;`
- `p = &arr[0];` → `p = arr`
- `p++ = &arr[1]` (int 형을 가르키기 때문에 사실 `p += 4`가 된다.)
- `p = arr`를 통해 `p++ = arr++??`
 - `arr++` 배열의 위치를 가르키는것이라 덧셈불가하다(`0x7fff1234`) ++;
 - 마찬가지로 `p != arr+4`이다
 - `p++` 후에 `*p = 5`하면 `arr[1] = 5`와 같은 효과를 지닌다.



포인터와 배열, 연산

```
char arr[3] = "abc";
```

```
char *arr2 = "abc";
```

arr[0] = '1'; 배열로 선언시 수정이 가능하지만

*(arr2) = '1'; 포인터로 선언해서 수정 불가능

```
int *p;
```

```
int num;
```

*p = 'A'; 그러나 배열이 아닌 경우에는 포인터선언도 수정 가능



포인터와 배열, 연산

```
void call_by_value(int a, int b)
```

```
{
```

```
    int temp;
```

```
    temp = a;
```

```
    a = b;
```

```
    b = temp;
```

```
}
```

```
void call_by_reference(int *a, int *b) call_by_reference(&a, &b); 주소값을 넘겨주면
```

```
{
```

```
    int temp;
```

```
    temp = *a;
```

```
    *a = *b;
```

```
    *b = temp;
```

```
}
```

call_by_value(a, b); 함수에서만 값이 바뀌고
main에서 호출시 a,b값 그대로

call_by_reference(&a, &b); 주소값을 넘겨주면
main에서 호출시 a,b값 바뀜



이중포인터

- `int ** pp;` 포인터를 가르키는 포인터

- `int *p;`

- `int a;`

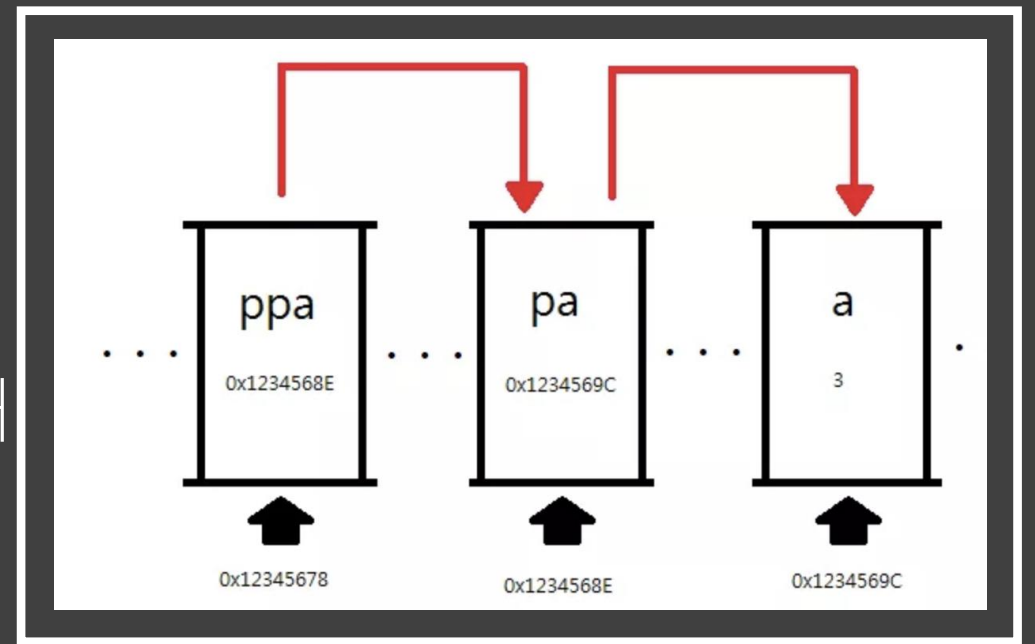
- `p = &a;`

- `pp = &p;`

- `int *****ppp;`이고

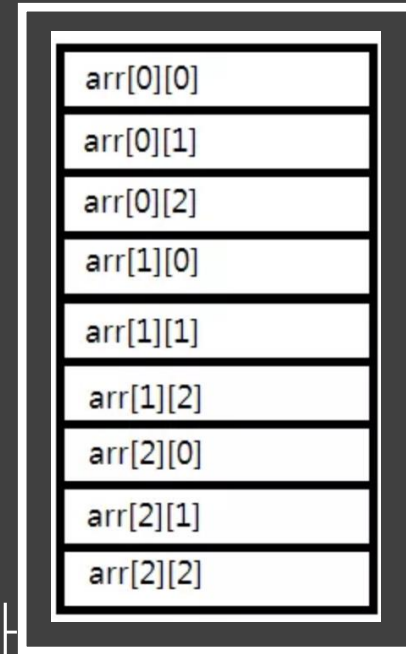
- `int *****pppp;`이면

- `pppp = &ppp;` 가능



이중포인터

- `int ** pp` 포인터를 가르키는 포인터
- 따라서 `int arr[2][3];`와 `int **p;`는 같을 수 없다
2차배열은 연속적인 값이기 때문이다
대신 `int (* p)[3]` 이런형태로는 사용 가능하다
- `int arr[2][3]; int brr[10][3]; int crr[2][5];`일 때
- `int (*p)[3];` `p = arr, p = brr, p != crr`만 불가능!



<code>arr[0][0]</code>
<code>arr[0][1]</code>
<code>arr[0][2]</code>
<code>arr[1][0]</code>
<code>arr[1][1]</code>
<code>arr[1][2]</code>
<code>arr[2][0]</code>
<code>arr[2][1]</code>
<code>arr[2][2]</code>



포인터 - 동적할당



이중포인터

- 힙이란
- Ram에 올라오는 프로그램 : 코드 / 데이터 segment
- 데이터 segment : 스택, 읽기전용 데이터, 힙등
- 데이터seg에서는 컴파일 할 때 할당할 크기가 정해져 있어야 하는데 힙메모리의 경우에만 사용자가 자유롭게 할당 및 해제를 할 수 있다.



malloc

- `#include <stdlib.h>` 라이브러리에 존재
- 할당받을 메모리 크기를 바이트단위로 받아서 할당.
할당이 완료된 경우 포인터를 반환
할당이 실패한 경우 널포인터를 반환
- 형태
(자료형/디폴트는 void)malloc(할당할 크기)



동적 할당 및 해제

- 1차배열 할당

```
int n;
```

```
scanf("%d", &n);
```

```
int *arr = (int *)malloc(sizeof(int)*n);
```

```
free(arr);
```



동적 할당 및 해제

- 2차배열 할당

```
int r, c;
```

```
scanf("%d, %d", &r, &c);
```

```
int **mat = (int **)malloc(sizeof(int*) * r);
```

 포인터를 인수로 갖는 이중배열 할당

```
for (int i = 0; i < r; i++)
```

```
    mat[i] = (int *)malloc(sizeof(int)*c);
```

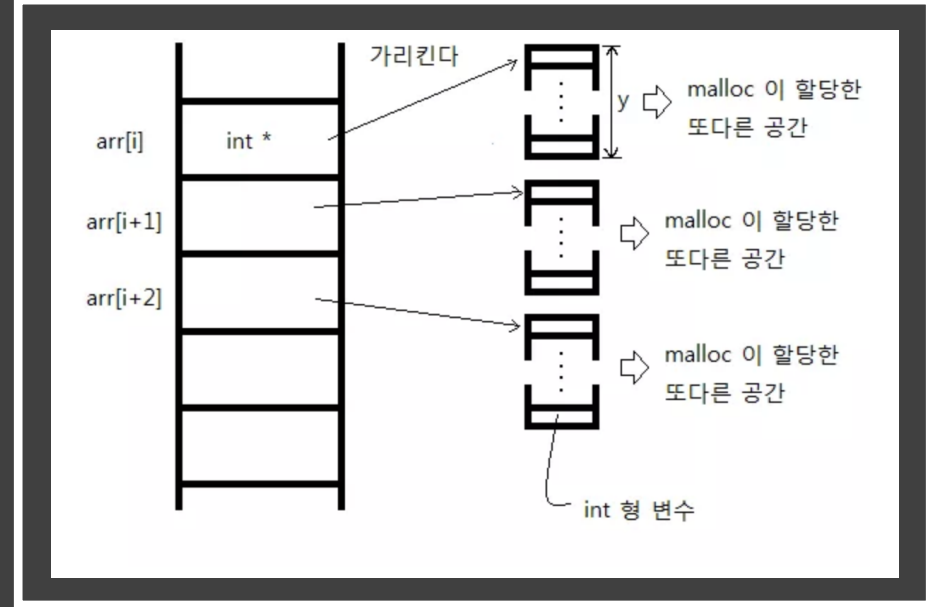
```
for (int i = 0; i < r; i++)
```

```
    free(mat[i]);
```

다.

```
free(mat);
```

따라서 할당 해제해 줄때도 for문을 통해 각각 free해줘야한



각 행마다 배열할당



이미지 출처

- <https://modoocode.com>



THANK YOU!

