Java를 알고 C배우기

컴퓨터프로그래밍3

week 11-3 동적 메모리-동적 메모리 좀더 알아보기

2022.1학기

충남대 조은선

널(NULL) 포인터

- malloc은 운영체제에게 메모리조각을 요청함
- 문제가 있어서, 가능한 heap 메모리가 확보가 안되면?
  - malloc은 NULL을 리턴
- Null을 활용한 안전한 프로그래밍 패턴

```
int* get_number( ){
   int* p;
   p = (int *)malloc(sizeof(int));
   if (p == NULL){
      printf("No more memory available.\n");
      exit(1);
   }
   *p = 20;
   return p;
}
```

calloc, realloc

- malloc 외에 메모리 동적 생성 방법

  calloc, realloc, xalloc, ... // 모두 free 필요

- calloc

  malloc과 동일. 단, 인자전달이 좀 다르고, 메모리를 0으로 초기화

```
int *p = (int*)calloc(24, sizeof(int));     // malloc(sizeof(int)*24)
```

- realloc

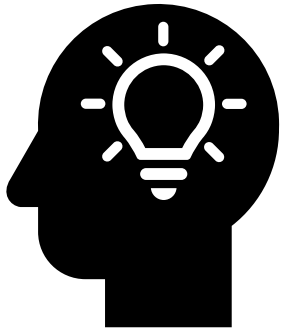
```
int *p = (int *)malloc(3 * sizeof(int));
int *new_p = (int *)realloc(p, 5 * sizeof(int));
```

  - 이전 포인터 p 주위의 공간이 넉넉하면: 주위에 좀더 메모리 확보 후 p를 그대로 리턴
  - 공간이 없을 경우: 이전 포인터 p가 가리키는 공간을 반납하고, 새로운 공간을 확보하고, 내용을 copy해서, 새로운 주소를 리턴
  - 일반적인 정적 배열을 나타내는 포인터에 대해서는 realloc 사용 불가

포인터와 배열 복습 (6-2)

▶ *(arr+i)는 arr[i]와 똑같다!

```
int main(void)
{
    int arr[3]={11, 22, 33};
    int * ptr=arr;
    printf("%d %d %d \n", *ptr, *(ptr+1), *(ptr+2));
    . . . .
}
```



배열이름도 포인터이니, 포인터 변수를 이용한 배열의 접근방식을 배열의 이름에도 사용할 수 있다. 그리고 배열의 이름을 이용한 접근방식도 포인터 변수를 대상으로 사용할 수 있다. 결론은 arr이 포인터 변수의 이름이건 배열의 이름이건

arr[i] == *(arr+i)

```
printf("%d %d %d \n", *(ptr+0), *(ptr+1), *(ptr+2));   // *(ptr+0)는 *ptr과 같다.
printf("%d %d %d \n", ptr[0], ptr[1], ptr[2]);
printf("%d %d %d \n", *(arr+0), *(arr+1), *(arr+2));   // *(arr+0)는 *arr과 같다.
printf("%d %d %d \n", arr[0], arr[1], arr[2]);
```

동적 변수를 배열처럼

- 여기서 배열 대신 동적 변수를 활용하면?

타입 잘 볼 것

```
int main(void)
{
    int * ptr = (int*) malloc(sizeof(int)*3));
    printf("%d %d %d \n", *ptr, *(ptr+1), *(ptr+2));
    . . . .
}
```

```
printf("%d %d %d \n", *(ptr+0), *(ptr+1), *(ptr+2));   // *(ptr+0)는 *ptr과 같다.
printf("%d %d %d \n", ptr[0], ptr[1], ptr[2]);
```

생성 이후로는, 앞페이지 포인터-배열 예와 다르지 않음

배열 형 동적 변수

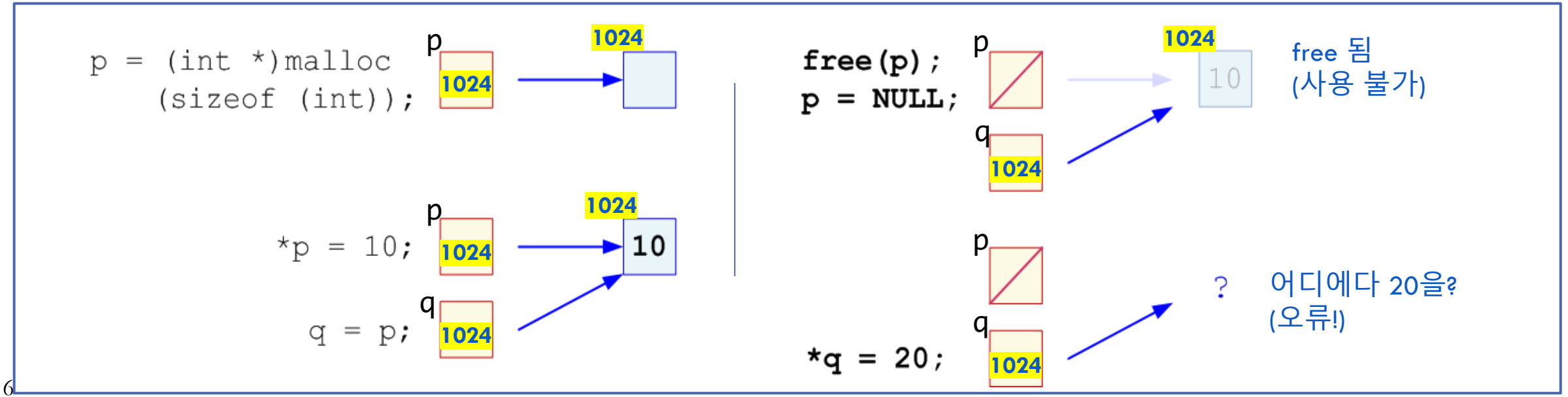
- 동적으로, 여러 개의 단일 타입 메모리들을 한꺼번에 모은 덩어리 메모리를 확보하고 싶을 때 사용
- 배열과 아주 비슷
- 왼쪽 예) int 요소가 3개인 메모리를 동적변수로 만들기
- **요소의 개수가 동적으로 정해질 때 매우 유용**
  - int arr[n]은 불가능
  - (int*) malloc((sizeof(int)*n)은 가능

포인터의 문제 1 - Dangling Pointer

```
int *p, *q;
p = (int*)malloc(sizeof(int));
*p = 10;        // 동적 변수 내용에 10을 넣음
q = p;          // 동적 변수 내용에 10을 넣음
free(p);
p = NULL;       // 명시적으로 NULL을 넣어주는 것이 좋음
*q = 20;        // 오류, q가 dangling pointer이다
```

Dangling pointer

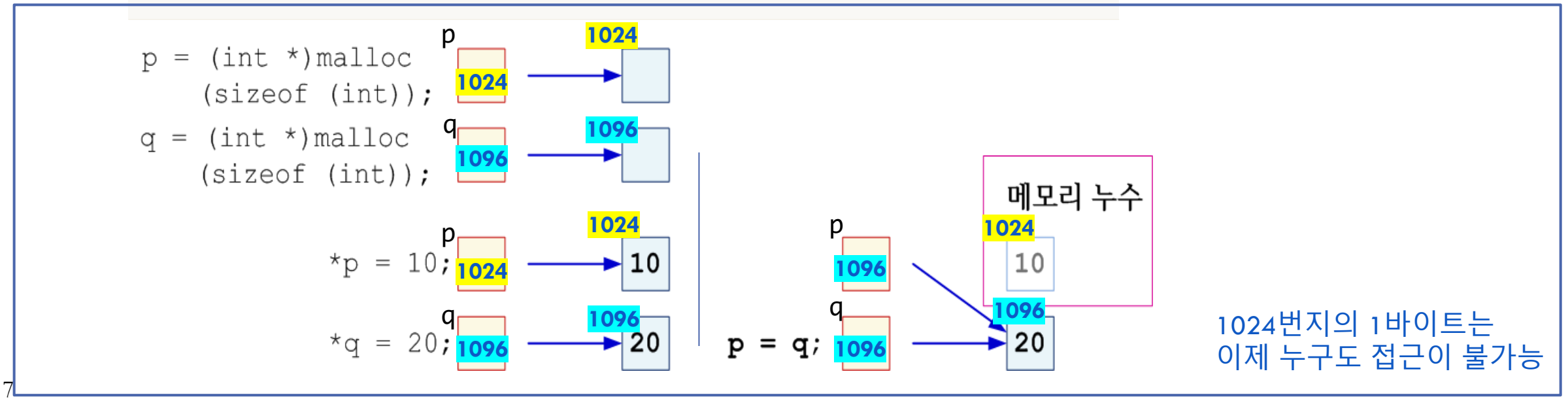
- 이미 free 된 메모리조각을 다른 포인터변수로 접근하게 되는 경우가 있음 (오류)
- 프로그램이 복잡해지면, 이런 경우가 매우 자주 생김 (조심)



포인터의 문제 2 – Memory Leak

- **Memory Leakage**
  - 공간을 반납하지도 않았고, 자기도 접근 불가

```
int* p = (int *)malloc(sizeof(int));
int* q = (int *)malloc(sizeof(int));
*p = 10; *q = 20;
p = q;  // 원래 p 공간은? 프로그램에서 접근할 방법이 없다
```



Quiz

다음과 같은 프로그램에 대한 설명 중 사실과 거리가 먼 것은? (답 3개)

```
int* p = (int *)malloc(sizeof(int));   //(ㄱ)
int* q = (int *)malloc(sizeof(int));   //(ㄴ)
int r = 30;                            //(ㄷ)
int *w = &r                            //(ㄹ)

*p = 10; *q = 20;                      //(ㅁ)
p = q;                                 //(ㅂ)
free(p);                               //(ㅅ)
w = (int *)malloc(sizeof(int));        //(ㅇ)
```

(1) (ㄹ)을 실행하고 나면 변수 w에 있는 주소의 내용은 30이 된다.
(2) (ㄹ)이후 변수 r의 값을 40으로 바꾸면 printf("%d", *w); 도 40을 출력한다.
(3) (ㅂ)을 실행하기 직전에 printf("%d", *p)를 실행하면 10을 출력한다
(4) (ㅂ)을 실행하고 나서 printf("%d", *p)를 실행하면 20을 출력한다
(5) (ㅂ)을 실행하고 나면 변수 p는 dangling pointer가 된다.
(6) (ㅅ)을 실행하고 나면 변수 q는 dangling pointer가 된다
(7) (ㅅ)을 실행하고 나면 (ㄱ)의 메모리는 접근이 불가능하다
(8) (ㅅ)을 실행하고 나면 (ㄴ)의 메모리는 접근이 불가능하다.
(9) (ㅇ)을 실행하고 나면 변수 w는 dangling pointer가 된다.
(10) (ㅇ)을 실행하고 나면 memory leak 이 발생한다