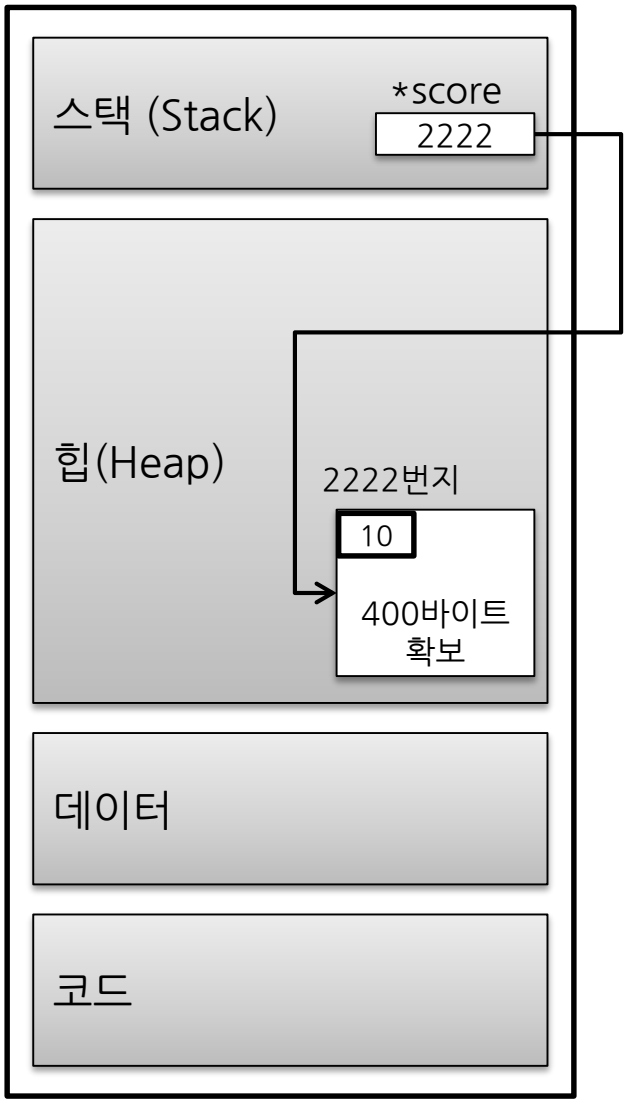
11차시 - 동적 할당(2)

동적 메모리 할당

- `void *malloc (size_t size)`
  - `size_t` : unsigned int 와 같다. 크기 지정.
  - `void *` : 범용 포인터를 반환
  - 할당 실패 : NULL 반환

```
int   *score;

score = malloc ( 400 );
score = malloc (100 * sizeof(int) );
        //좀 더 안전
score[0] = 10;
```



동적 할당의 예

```
int    size;

printf("몇 개의 배열이 필요합니까?");
scanf("%d", &n);
int  num[n];
```

(X)

```
int    size;
int    *num;

printf("몇 개의 배열이 필요합니까?");
scanf("%d", &size);
num = malloc(size * sizeof(int));
```

(O)

- 할당 후에는 자유롭게 사용

```
num[ i ] = 0;

*( num + i ) = 0;
```

할당 실패

- **메모리 할당은 실패할 수 있다.**
  - 예외 처리 필수

```
int     *value;

value = malloc(100 * sizeof(int));

if (value == NULL) {
    printf("메모리 확보 실패\n");
    exit(0);    // 프로그램 종료
}

*value = 100;
```

메모리 할당시 주의점

- 문자열의 널 문자를 담을 공간을 고려하자.

```
p = malloc(strlen(str)); // 가 아니라 +1을 해야 한다.
```

- 할당한 크기를 넘는 사용은 **비정상 종료**
- 할당된 공간은 자동 초기화되지 않는다(**쓰레기값**)

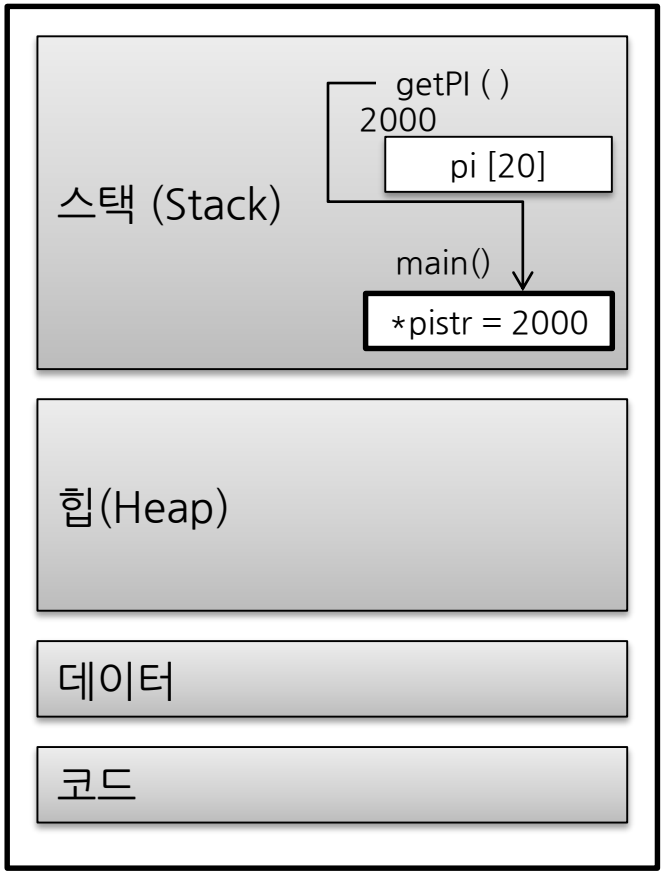
달려보자

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        char  *mblock1, *mblock2;
        int   i;

        printf("메모리 할당을 시작합니다.");
        getchar();

        mblock1 = malloc(100000000);      // 100MB
        for (i = 0; i < 100000000; i++) *(mblock1 + i) = 0;
        printf("100MB를 할당했습니다.");
        getchar();

        mblock2 = malloc(1000000000);     // 1 GB
        for (i = 0; i < 1000000000; i++) *(mblock2 + i) = 0;
        printf("추가로 1GB를 할당했습니다.");
        getchar();

        return 0;
}
```

작업관리자

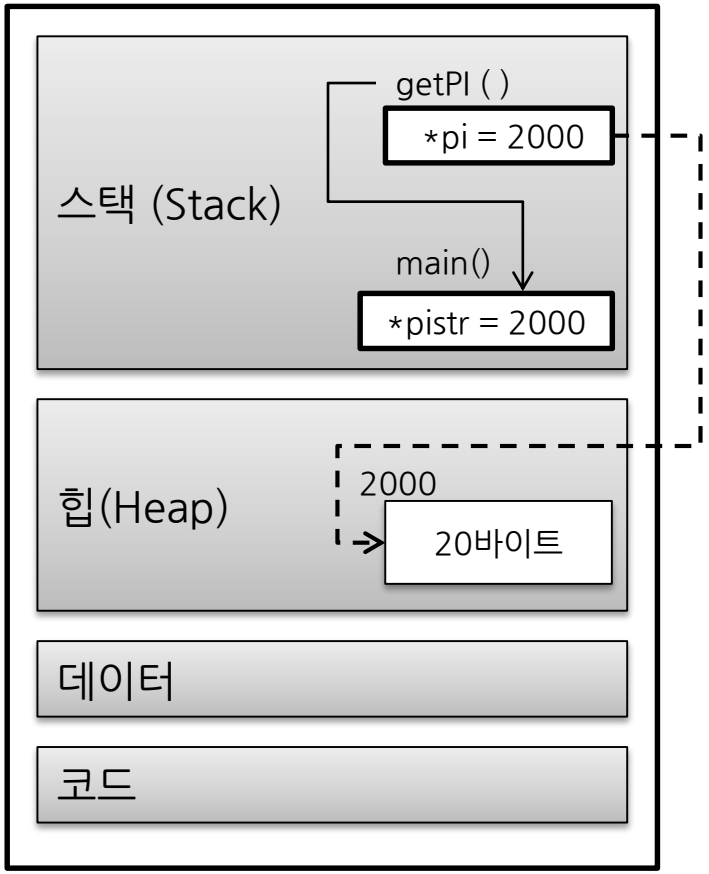
메모리 공유

```
char *getPI(void)
{
    char PI[20];

    strcpy(PI, "3.1415926535897932");
    return PI;
}


int main(void)
{
    char  *pistr;

    pistr = getPI();
    printf("%s\n", pistr);

    return 0;
}
```



메모리 공유

```
char *getPI(void)
{
    char *PI;

    PI = malloc(20);

    strcpy(PI, "3.1415926535897932");
    return PI;
}

int main(void)
{
    char  *pistr;

    pistr = getPI(); //20바이트 할당하고 반환
    printf("%s\n", pistr);

    return 0;
}
```



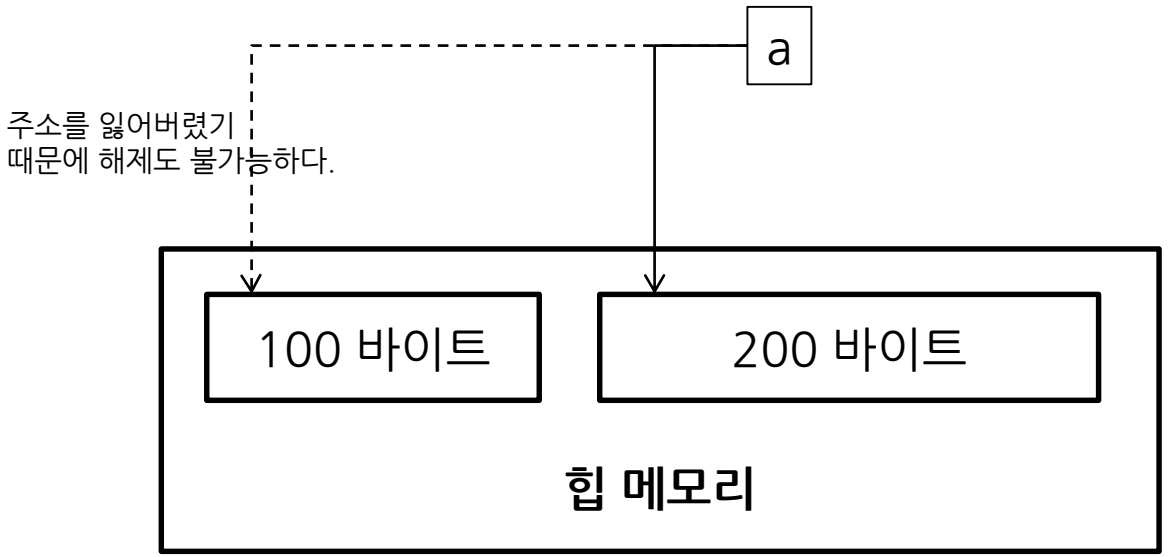
메모리 해제

- `void free(void *p)`
  - p : 해제할 메모리의 포인터 (malloc의 반환값)

- **주의점**
  - **해제는 전체만** 가능(일부 반납 불가)
  - **잘못된 메모리 해제는 비정상 종료**

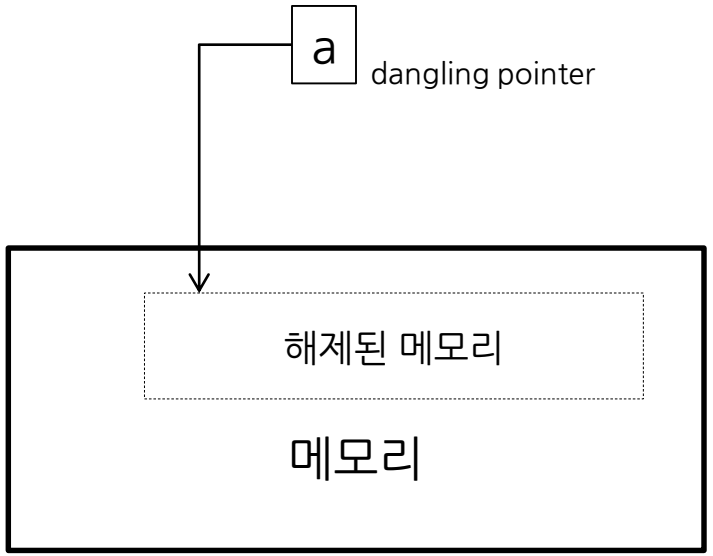
메모리 해제

```
a = malloc(100);
a = malloc(200);

free(a);
```



댕글링 포인터(dangling pointer)

- **free한 후에도 포인터는 계속 그 위치를 가리킨다.**
  - 이것을 댕글링(dangling) 포인터라 한다.
  - 이 포인터를 써서는 안 된다(비정상 종료).

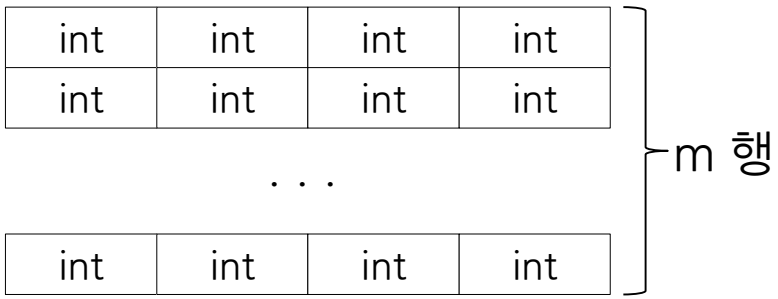


대처법은 뒤에서

다차원 공간의 할당

- 메모리는 어차피 1차원이다.
- 할당도 1차원이다.
  - 따라서 1차원으로 변환하여 생각하자.

```
int   *data;

scanf("%d", &m);
data = malloc(sizeof(int)*4*m);

for(i=0;i<m;i++)
   for(j=0;j<4;j++)
      scanf("%d", data+i*4+j);
      // scanf("%d",&data[i][j]);
      // 로 쓰고 싶지만,,,
```



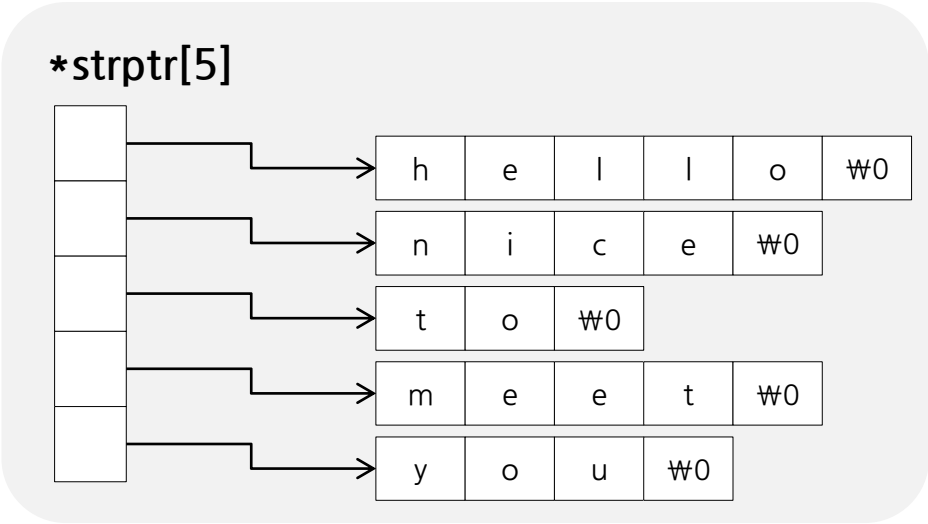
다차원 공간의 할당

- scanf("%d", &data[i][j]); **로 쓰고 싶지만,,,**
  - 규격이 없어 주소 계산이 안 된다.
  - 규격을 준다면?

```
int   (*data)[4];

scanf("%d", &m);
data = malloc(sizeof(int) * 4 * m);

for(i = 0;i < m;i++)
   for(j = 0;j < 4;j++)
      scanf("%d", &data[i][j] );
```

문자열의 메모리 절약

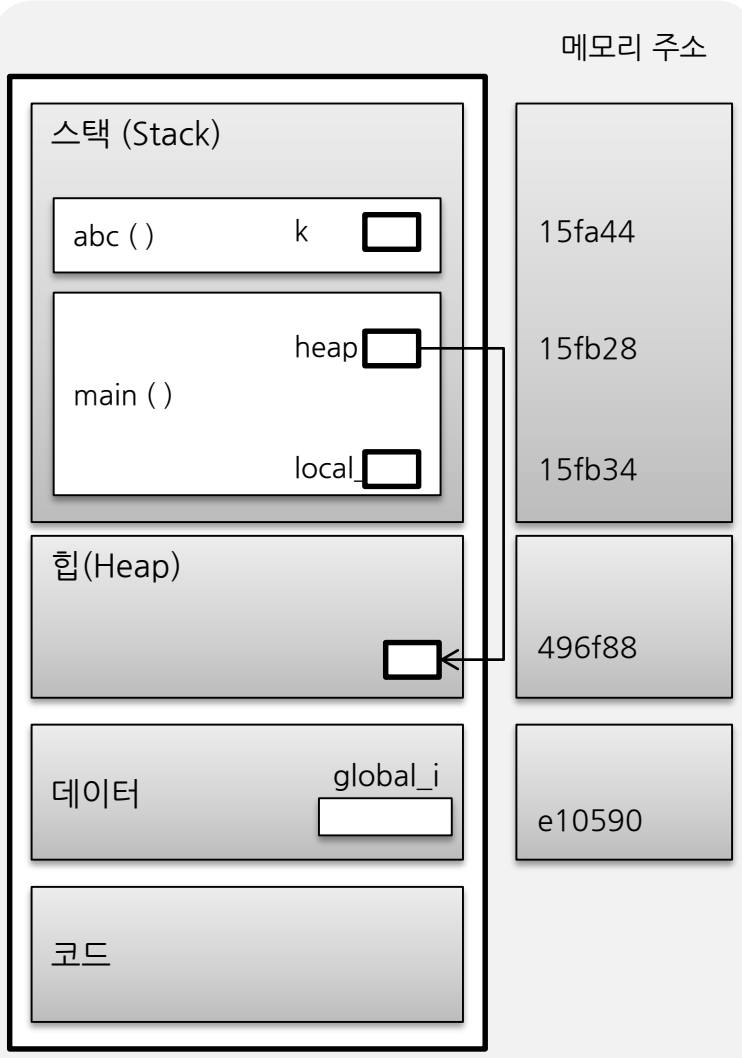
```
int i;
char input[100];                    // 충분한 크기의 입력을 위해 100바이트로 정의
char *strptr[5];

for (i = 0;i < 5; i++) {
        gets(input);
        strptr[i] = malloc(strlen(input) + 1);
        strcpy(strptr[i], input);
}

for (i = 0;i < 5; i++)
        printf("%s\n", strptr[i]);
```



배열과 비교해 보자.

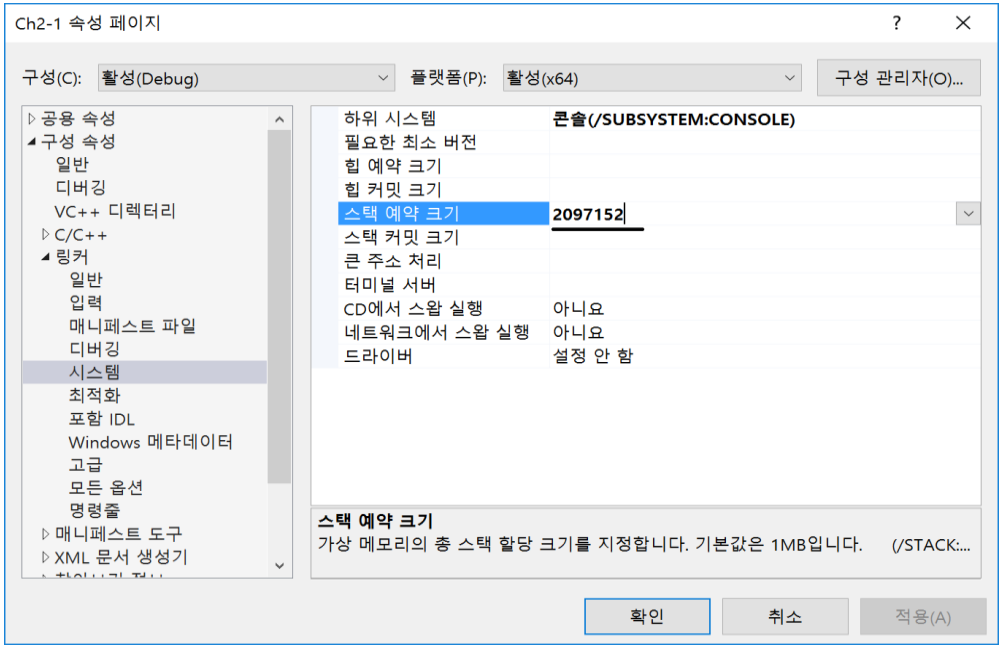
메모리 구조의 확인

```
int   global_i;

int main(void) {
   int   local_i;
   int   *heap ;

   heap = malloc(4);

   printf("global variable : %x\n", &global_i);
   printf("local variable : %x\n", &local_i);
   printf("local variable : %x\n", &heap);
   printf("heap memory variable : %x\n", heap);

   abc();
   return 0;
}

int  abc(void)
{
   int k;

   printf("stack memory variable : %x\n", &k);
}
```



스택 크기의 결정

- **스택의 크기는 결정되어 있다.**
  - 컴파일러에 의해 결정
  - VS 에서는 1 MB
- **옵션에서 변경할 수 있다.**
- **변경할 경우 위험하다.**
  - 이유는?



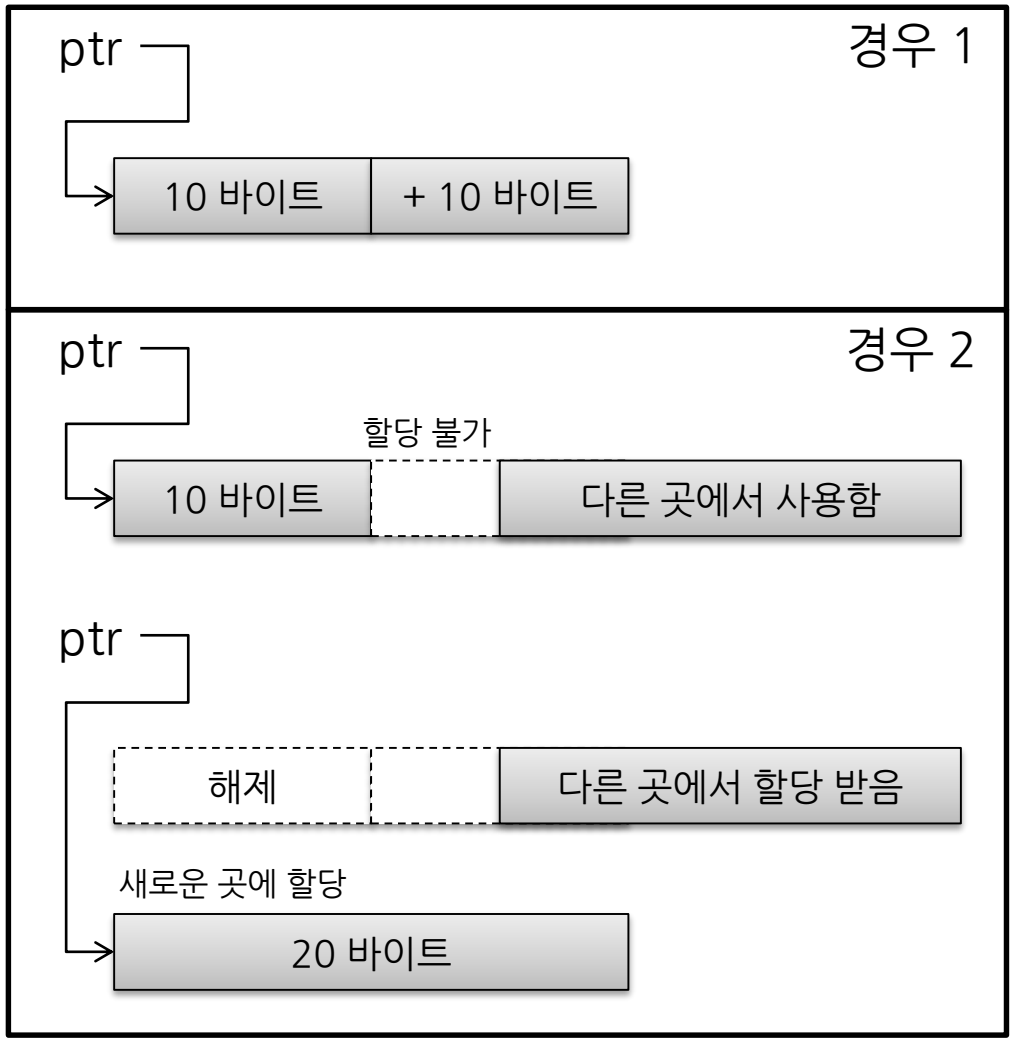
메모리 함수

- calloc
  - malloc과 동일하되, 0으로 초기화를 진행한다.
  - `void* calloc (size_t num, size_t size);`
  - `char *str = calloc ( 100, sizeof( char ) );`
- realloc
  - 할당 받은 메모리의 크기를 바꾼다.(재할당)
  - 커질 수도, 작아질 수도 있다.
  - `void* realloc (void* ptr, size_t size);`

realloc

```
ptr = malloc(10);
…
ptr = realloc(ptr, 20);
```

- **주의사항**
  - 초기화 없음
  - 할당 실패할 수도 있음
  - 첫 인자가 NULL이면 malloc과 동일
  - size가 0이면 free와 동일
- **단점은?**



그 외의 메모리 함수

- memset
- memcpy
- memmove
- memcmp
- memchr

여러분이 만드는 것보다 속도가 빠르다.