C Programming (CSE2035)
(Chap9. Pointer Applications 3)

Sungwon Jung, Ph.D.

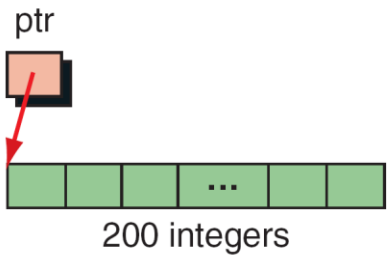
Bigdata Processing & DB LAB
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

Contiguous Memory Allocation - calloc()

- 두 번째 메모리 관리 함수는 **calloc()** 이다.
  - calloc() 함수의 프로토타입은 다음과 같다.

```
void *calloc (size_t element-count,
              size_t element_size);
```

  - 할당 받은 메모리를 사용할 하나의 원소 크기와 전체 원소의 개수를 개별적인 인자로 받는다 .
  - calloc() 함수는 메모리를 할당하면서 해당 메모리를 0으로 **초기화** 시킨다.




```
if (!(ptr = (int*)calloc (200, sizeof(int))))
    // No memory available
    exit (100) ;

// Memory available
…
```

malloc()과 calloc()의 차이점

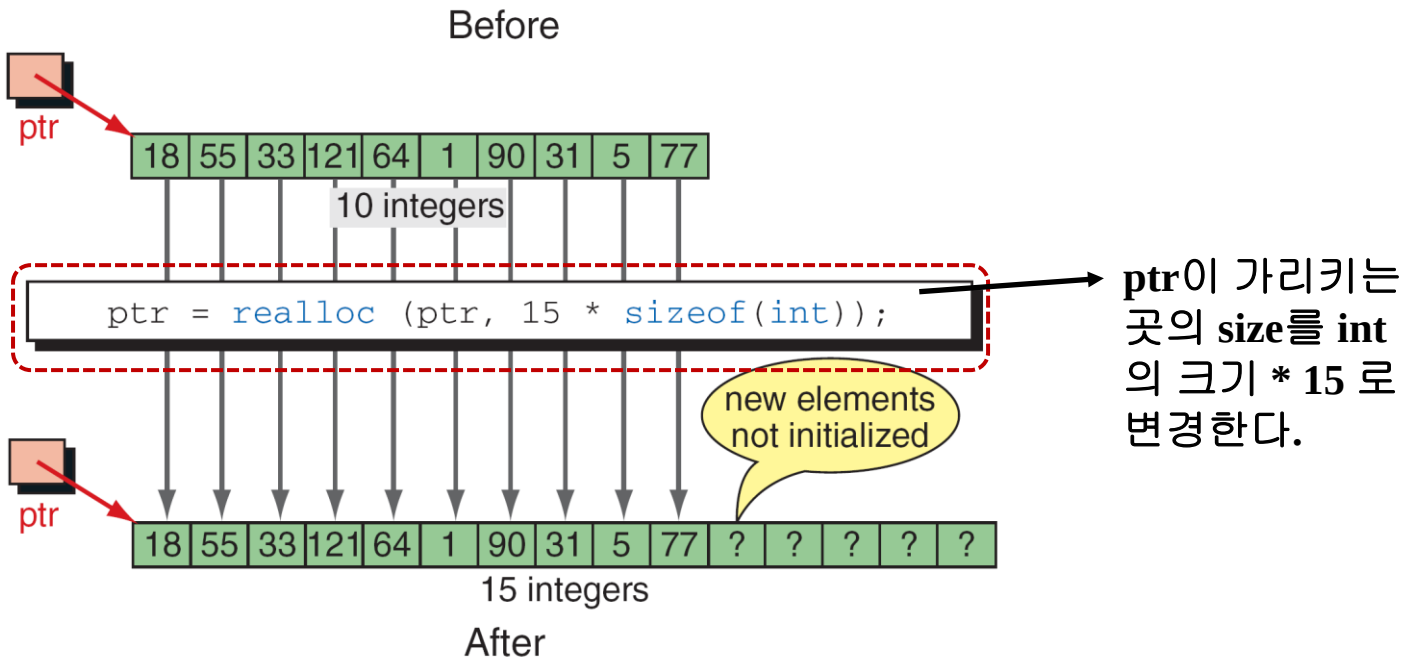
<calloc함수 호출 예>

Reallocation of Memory - realloc ()

- **realloc()** 함수는 변수의 메모리를 동적으로 변경하기 위하여 사용한다.
- 함수의 프로토타입은 다음과 같다.

```
void *realloc (void *ptr, size_t newSize);
```

- 즉, **ptr**이 현재 할당하고 있는 메모리의 크기를 **newSize**로 변경한다.



calloc() & realloc()

- 예제 프로그램 - **realloc**함수를 이용한 양수값 입력 프로그램

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

int main(void)
{
        int *ip;
        int size = 5;
        int cnt = 0;
        int num;
        int i;

        ip=(int *)calloc(size, sizeof(int));
        while(1){
                printf("Input the positive integer : ");
                scanf("%d", &num);
                if(num<=0) break;
                if(cnt<size) {
                        ip[cnt++]=num;
                }
                else {
                        size += 5;
                        ip=(int *)realloc(ip, size*sizeof(int));
                        ip[cnt++]=num;
                }
        }
        for(i=0; i<cnt; i++) {
                printf("%5d", ip[i]);
        }
        printf("\n");
        free(ip);
        return 0;
}
```

일정한 크기의 기억공간을 할당(calloc) 받고 나서, 입력되는 데이터가 양수일 경우에 기억공간을 재할당(realloc)한다.

할당 받은 기억공간이 남아있으면 데이터를 저장한다.

기억공간이 부족하면 크기를 늘려서 재할당 받는다.

```
[root@mclab chap10]# vi chap10-7.c
[root@mclab chap10]# gcc -o chap10-7 chap10-7.c
[root@mclab chap10]# ./chap10-7
Input the positive integer : 36
Input the positive integer : 28
Input the positive integer : 14
Input the positive integer : 0
   36   28   14
[root@mclab chap10]#
```

Releasing Memory - free ()

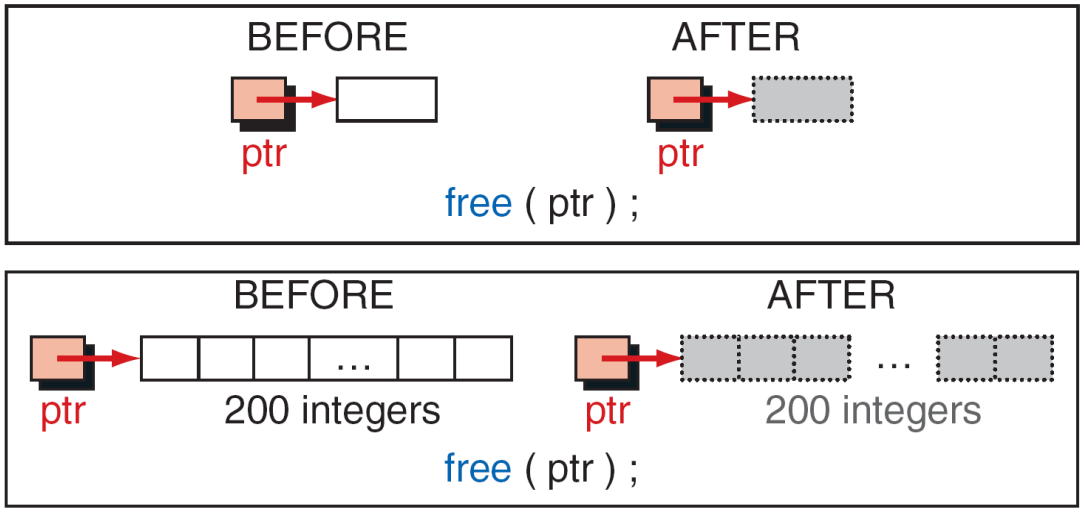
- 동적으로 할당 받은 메모리 영역에 대하여 반환하는 함수
  - free() 함수의 프로토타입은 다음과 같다.

```
void free (void *ptr);
```

  - 사용하지 않는 메모리에 대한 계속적인 할당은 메모리 사용의 낭비의 단점이 있기 때문에 동적으로 할당 받은 필요 없는 메모리 영역에 대해서는 free()를 통해서 메모리를 반환하도록 한다.
  - free() 는 ptr이 가리키는 기억 장소를 해제한다.
  - 단, ptr이 NULL일 때는 아무 일도 하지 않는다.
  - ptr은 malloc(), calloc(), realloc()에 의하여 이전에 할당되었던 기억 장소의 포인터이어야 한다.

Releasing Memory - free ()

- 다음은 **free()**의 동작을 보여주는 그림이다.



- **Caution!**

동적으로 할당한 기억 장소 영역은 책임지고 **해제**하지 않으면 안 된다. 영역을 해제할 때까지는 반드시 포인터를 기억해 두어야 한다.

Releasing Memory - free ()

- 예제프로그램 - 정수를 1개 할당, 해제하는 프로그램

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void main(void)
5  {
6          int *p;
7          p = (int *)malloc(1*sizeof(int));
8          if(p==NULL)
9                  printf("Not allocated!\n");
10         else
11                 free(p);
12 }
```

메모리 4byte를 할당 받음

제대로 할당되었는지 확인

메모리 4byte를 해제

Memory leak

- 더 이상 사용되지 않는 메모리 공간의 반환
  - 동적으로 할당 받은 메모리 공간은 더 이상 해당 공간을 참조하는 포인터가 없는 경우에도 heap영역에 자리를 차지하고 있다.
  - 동적 할당으로 생성된 메모리를 적절한 타이밍에 해제하지 않으면 해당 메모리는 생성된 만큼의 공간을 차지하면서도 정작 사용할 수 없는 상태가 된다

```
int *intArr;
intArr=(int *)malloc(sizeof(int)*5);
intArr=(int *)malloc(sizeof(int)*10);
free(intArr);
```

프로그램이 종료될 때 까지 남아있으나 사용할 수는 없는 메모리

  - 메모리 누수가 반복적으로 일어날 경우 시간이 지나면서 사용할 수 있는 메모리를 모두 사용하여 프로그램이 더 이상 올바르게 기능하지 못하게 된다.
  - 이와 같은 문제를 메모리 누수(memory leak)라고 한다.

Dangling pointer

- **Dangling pointer**
  - 메모리가 삭제되거나 재 할당 될 때 유효하지 않은 메모리 공간을 참조하는 포인터.
  - 해제된 공간을 참조하고 있거나 지역변수를 참조할 때 발생하기 쉽다.

```
int main(){
    int *intPointer;
    dangle(&intPointer);
    printf("%d", *intPointer);
    return 0;
}
void dangle(int **intPtr){
    int a=1;
    *intPtr=&a;
}
```

함수의 지역변수를 참조하고 있지만 함수 수행 종료 후 해당 변수는 **stack**에서 제거되기 때문에 **dangling pointer**가 된다

Dangling pointer

```
int main(){
    int *intPointer;
    intPointer=(int*)malloc(sizeof(int)*5);

    for(int i=0;i<5;i++)
        intPointer[i]=i;

    dangle(intPointer);

    printf("%d ", intPointer[1]);

    return 0;

}
void dangle(int *intPtr){
    free(intPtr);
}
```

이미 해제된 메모리 공간의 주소를 그대로 참조하고 사용하게 되면 프로그램이 오작동 하게 된다.

- Dangling pointer에 대한 접근은 프로그램의 오작동이나 중단을 유발하지만 컴파일 단계에서 검출되지 않기 때문에 주의가 필요하다.

Array of pointers

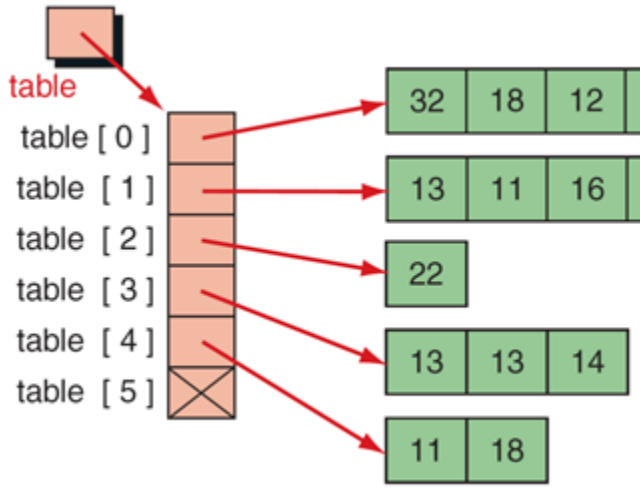
- 여러 개의 배열을 처리할 때, 포인터 변수가 배열의 개수만큼 필요
  - 하나의 Pointer는 하나의 Array에 대응
  - Pointer도 하나의 타입이므로 Pointer Array도 존재한다
  - 즉, Pointer Array를 사용하여 2차원 배열을 다룰 수 있다

32	18	12	24			
13	11	16	12	42	19	14
22						
13	13	14				
11	18					

  - 위의 데이터를 저장하기 위해 2차원 배열(5×7)을 이용한다면 메모리의 많은 낭비를 초래한다.

Array of pointers

- 이전의 2차원 배열을 포인터 배열로 나타내면 다음과 같다.




```
int *table[6];

table[0] = (int *)malloc(sizeof(int) * 4);
table[1] = (int *)malloc(sizeof(int) * 7);
table[2] = (int *)malloc(sizeof(int) * 1);
table[3] = (int *)malloc(sizeof(int) * 3);
table[4] = (int *)malloc(sizeof(int) * 2);
table[5] = NULL;

table[0][0] = 32;    table[0][1] = 18;
table[0][2] = 12;    table[0][3] = 24;
```

Array of pointers

- 예제프로그램
  - **M×N 2차원 matrix**를 포인터 배열을 이용하여 동적으로 생성하는 프로그램

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

void printMatrix(int *arr, int size);

void main( ){
    int m, n;
    int **matrix;
    int i, j;

    printf("Number of Rows : ");
    scanf("%d", &m);
    printf("Number of Cols : ");
    scanf("%d", &n);

    matrix = (int **) malloc (sizeof(int *) * m);
    for(i=0; i<m; i++)
        matrix[i] = (int *) malloc (sizeof(int) * n);
```

배열 출력 함수 선언

배열의 **row**의 개수 입력

배열의 **column**의 개수 입력

포인터 배열의 동적 할당

각 포인터에 동적으로 1차원 배열 할당

Array of pointers

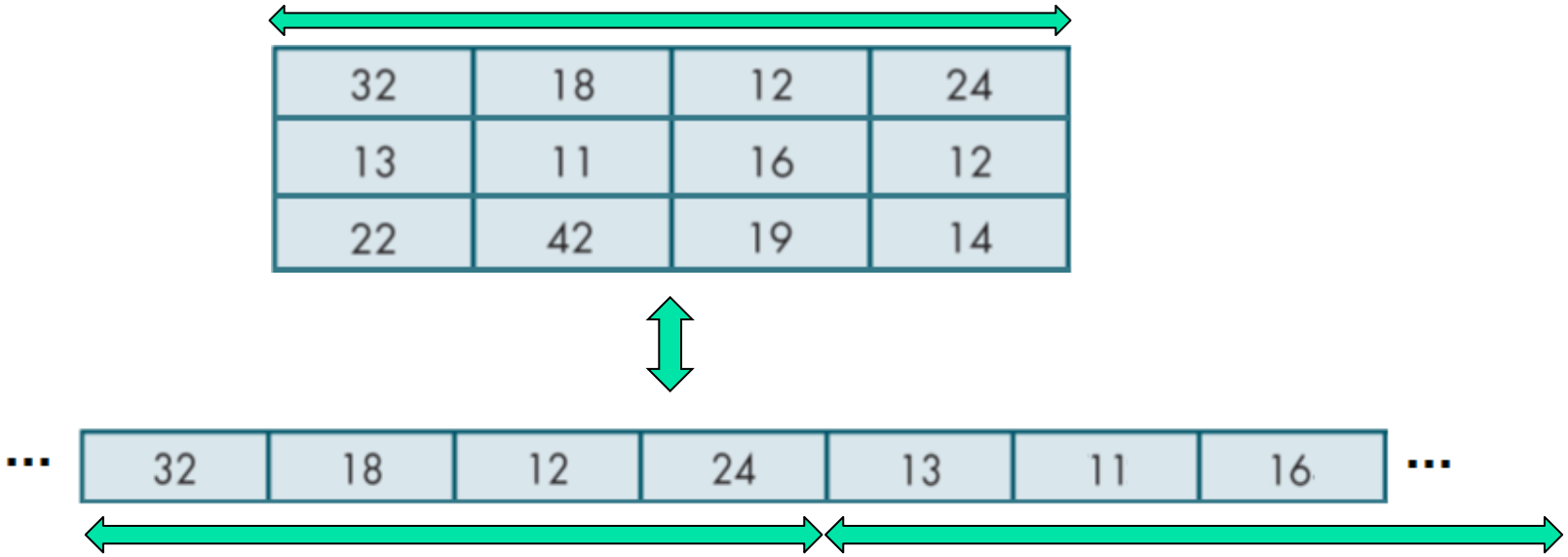
```
    for(i=0; i<m; i++)
        for(j=0; j<n; j++)
            matrix[i][j] = (i+1)*(j+1);

    for(i=0; i<m; i++)
        printMatrix(matrix[i], n);
}


void printMatrix(int *arr, int size){
    int i;

    for(i=0; i<size; i++)
        printf("%d ", arr[i]);
    printf("₩n");
}
```

한 라인씩 출력

```
Number of Rows : 4
Number of Cols : 9
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
```

Array of pointers

- **2차원 Array와 1차원 Array의 관계**
  - 실제 Memory는 1차원의 연속된 공간으로 이루어져 있음
  - 즉, 2차원 Array도 실제 Memory상에서는 연속적으로 저장되어 있다



  - Array[1][2] = 16에 해당하는 1차원 배열은 Array[6] = Array[1×4+2]에 해당하는 것을 알 수 있다

Array of pointers

- 예제프로그램
  - 1차원 배열로 나타내진 **m*n matrix**를 2차원으로 관리하는 프로그램

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

void main( ){
    int m, n, i, j;
    int *oneArray, **twoArray;

    printf("Number of Rows : ");
    scanf("%d", &m);
    printf("Number of Cols : ");
    scanf("%d", &n);

    oneArray = (int *)malloc(sizeof(int) * (m * n));
    twoArray = (int **)malloc(sizeof(int*) * m);

    for(i=0; i<m*n; i++) oneArray[i] = i+1;
    for(i=0; i<m; i++)
        twoArray[i] = &oneArray[i*n];
```

배열의 **row**의 개수 입력

배열의 **column**의 개수 입력

M*N matrix의 동적 할당

M*N matrix의 각 **row**에 해당하는 시작 주소들을 저장

Array of pointers

```
    for(i=0; i<m*n; i++){
        printf("%d ", oneArray[i]);
        if((i+1)%n == 0) printf("₩n");
    }

    for(i=0; i<m; i++){
        for(j=0; j<n; j++)
            printf("%d ", twoArray[i][j]);

        printf("₩n");
    }
}
```

1차원 Array를 2차원 처럼 출력하는 방법

2차원 Array표현

```
Number of Rows : 5
Number of Cols : 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
17 18 19 20
```

Segmentation Fault

- **Segmentation Fault (Segfault)**
  - 프로그램이 허용되지 않은 메모리에 접근할 때 발생한다
  - 주로 NULL로 설정된 영역에 값을 쓴다던가, 할당 받은 메모리를 벗어난 곳을 참조할 때 발생하기 쉽다

```
int *p = NULL;
*p = 3;
```

```
int *q = (int *)malloc(sizeof(int) * 4);
*(q+4) = 3;
```

  - Pointer가 실제 메모리에서 어디를 가리키고 있는지는 프로그램이 실행되기 전까지 알 수 없기 때문에, 메모리의 중요한 부분을 보호하기 위해 발생하는 에러이다

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

scanf("%d", &pos);
printf("%d", arr[pos]);
```

pos의 값이 arr가 할당된 범위를 벗어나는지 아닌지의 여부는 프로그램 실행 전까지 알 수 없다

Segmentation Fault

- **Segmentation Fault (Segfault)**
  - 따라서, Pointer를 사용할 때는 값이 유효한지 반드시 확인하고 사용해야 한다

```
if( p == NULL )
    printf("Pointer is null₩n");
else
    printf("%d₩n", *p);
```

  - 배열을 동적으로 할당 받아서 사용 중일 때는 반드시 index가 유효한 범위인지 확인하고 사용하여야 한다

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

if(idx >= n)
    printf("out of index");
else
    printf("%d₩n", arr[idx]);
```

  - Segmentation Fault가 발생하였다면, 이 두 가지를 우선적으로 생각해보아야 한다