C언어 (CSE2035)
(Chap9. Pointer Applications) (3-2)

Sungwon Jung, Ph.D.

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

Array of pointers

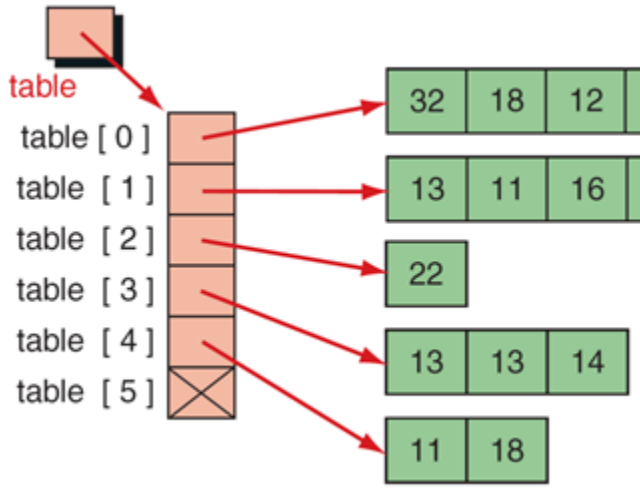
- 여러 개의 배열을 처리할 때, 포인터 변수가 배열의 개수만큼 필요
  - 하나의 Pointer는 하나의 Array에 대응
  - Pointer도 하나의 타입이므로 Pointer Array도 존재한다
  - 즉, Pointer Array를 사용하여 2차원 배열을 다룰 수 있다

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 matrix를 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 row의 개수 입력

배열의 column의 개수 입력

포인터 배열의 동적 할당

각 포인터에 동적으로 1차원 배열 할당

Array of pointers

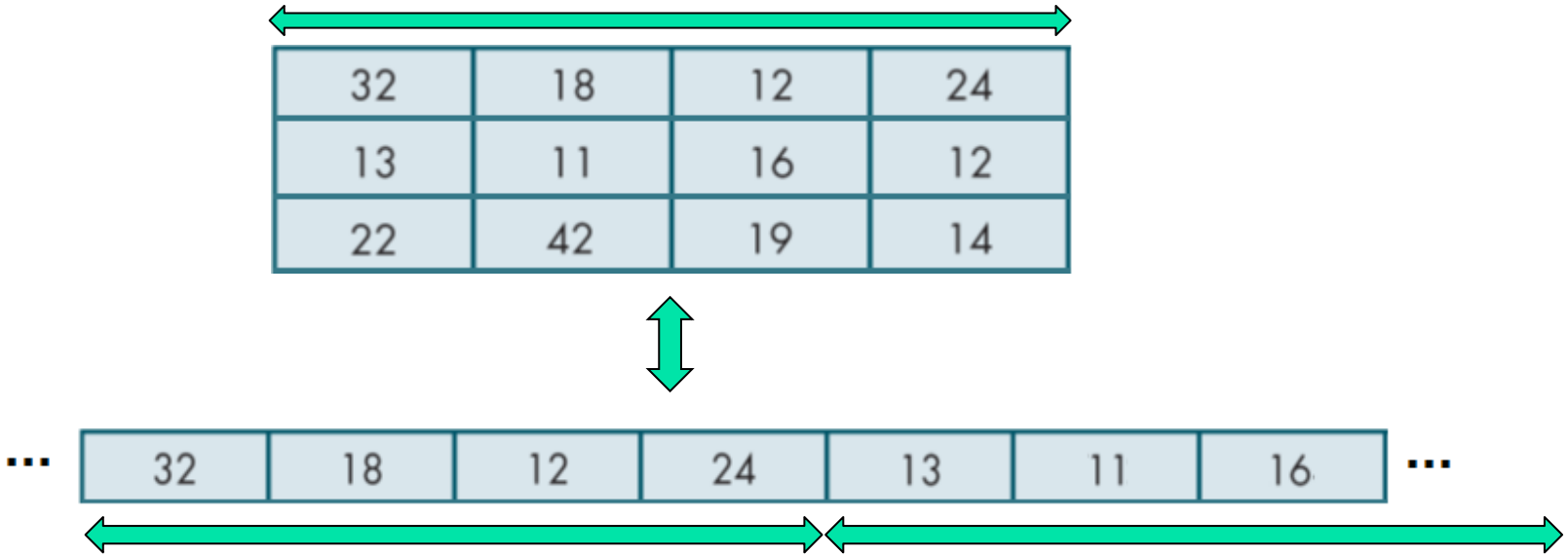
```
    for(i=0; i<m; i++)
        for(j=0; j<n; j++)
            matrix[i][j] = (i+1)*(j+1);

    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 matrix를 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 row의 개수 입력

배열의 column의 개수 입력

M*N matrix의 동적 할당

M*N matrix의 각 row에 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