전산 SMP 8주차

2014. 11. 15

김범수

bskim45@gmail.com

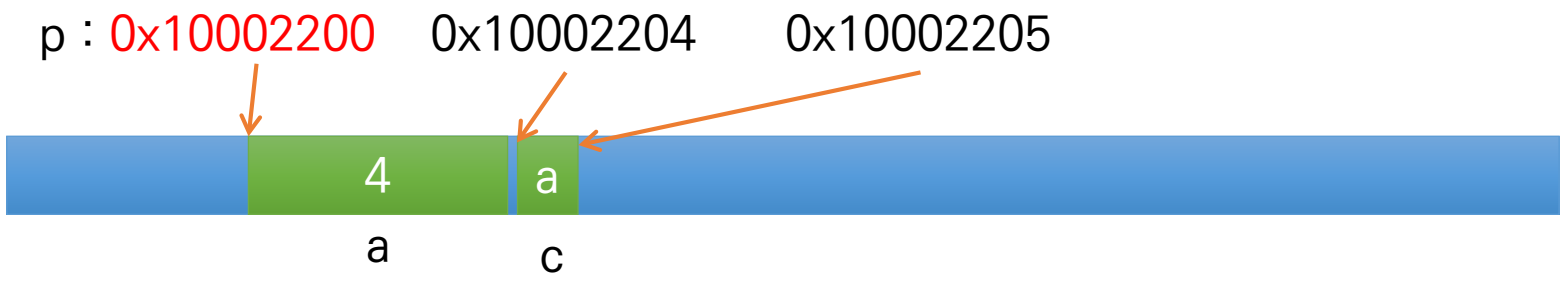
Special thanks to 박기석 (kisuk0521@gmail.com)

지난 내용 복습

Searching, Array Pointer

Pointer란?

- 데이터 접근에 사용되는 주소를 저장하는 타입 – 포인터도 타입이다
- int a = 4; char c = ‘a’; int *p = &a;



- 메모리는 긴 일차원 공간으로 볼 수 있고, 각 byte는 자신만의 주소를 가지고 있다.

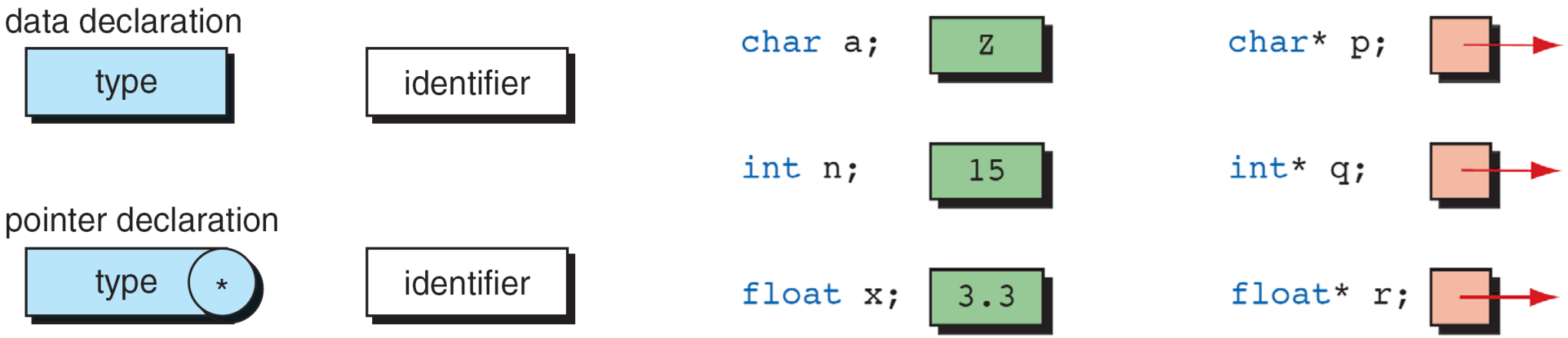
Pointer 선언, 사용

- Declaration

  int *ptr;

  → 정수형 데이터가 저장된 메모리의 위치(주소)를 저장할 수있는 포인터 변수

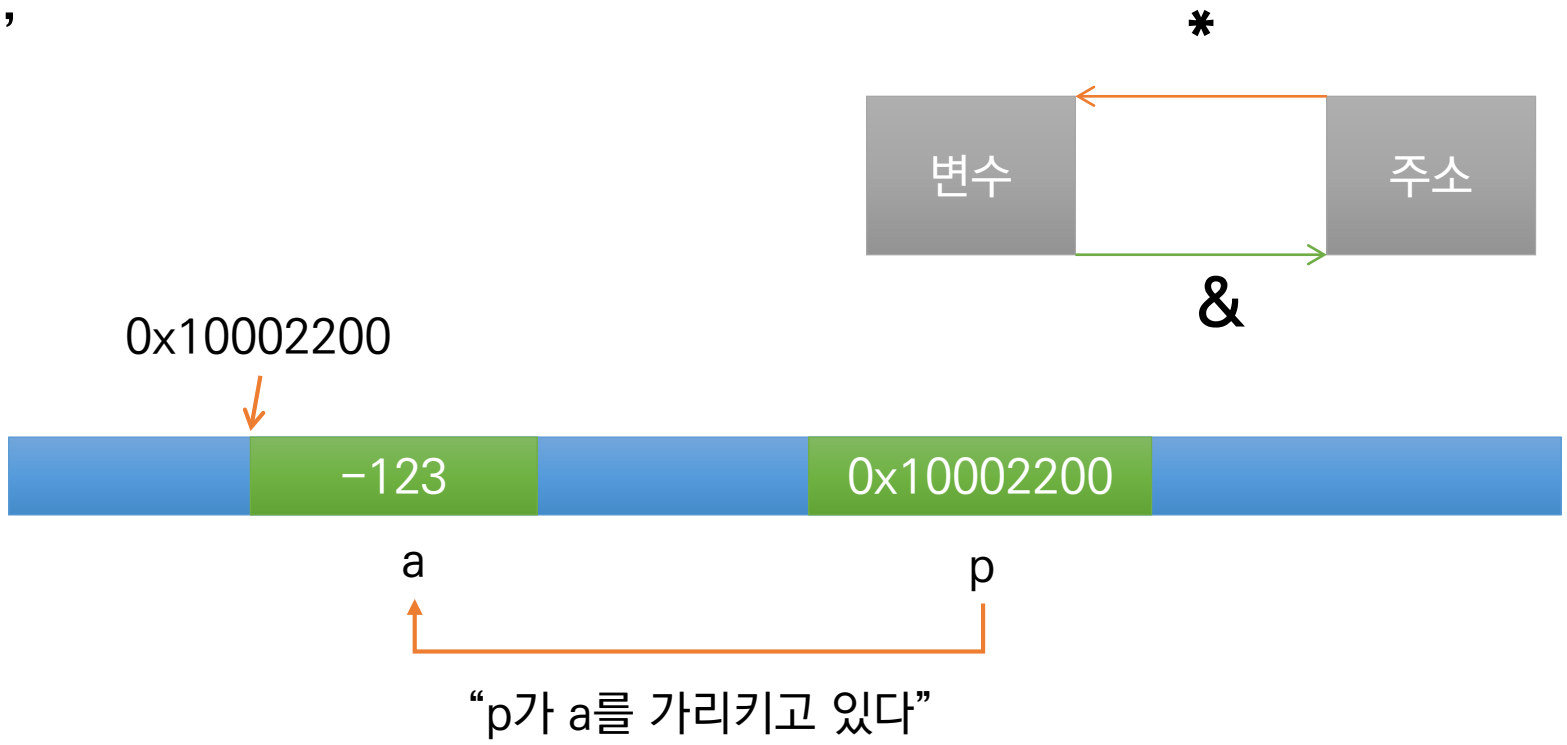
- 포인터의 타입 크기
  - 4 byte!(32 bit) ex) int : 4byte, char : 1 byte



Pointer 선언, 사용

int a = −123;

int *p = &a;



포인터 변수도 결국 ‘변수’이기 때문에 자신이 가리키는 주소를 메모리 어딘가에 저장하고 있다.

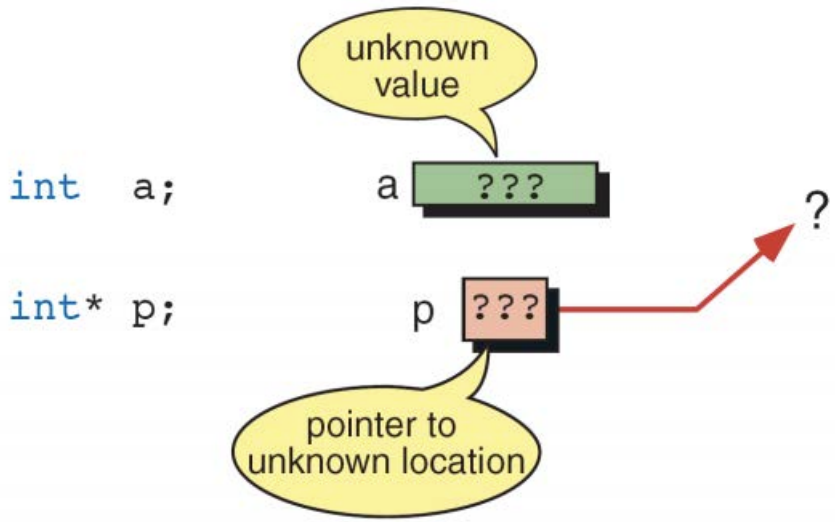
예제

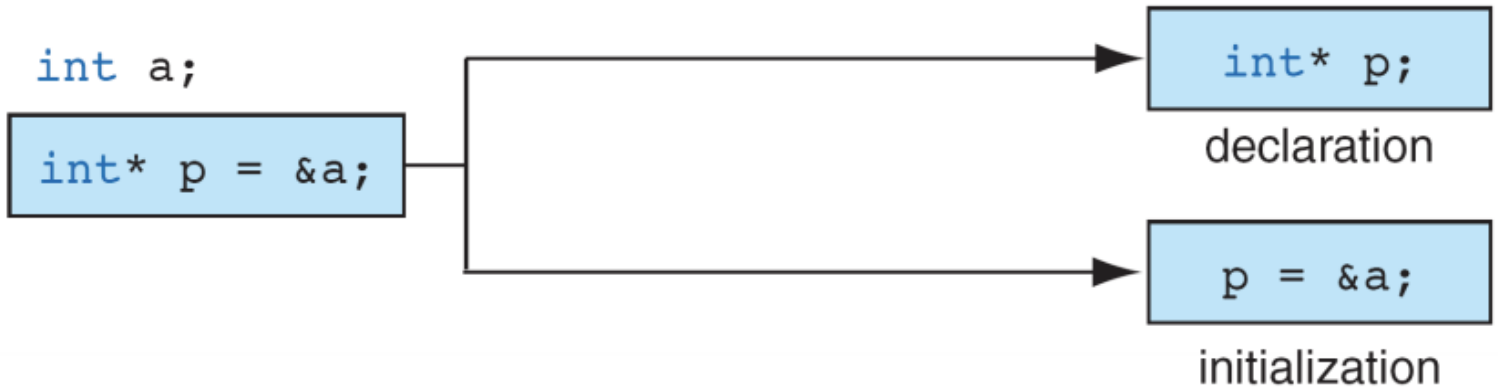
```
int a = 14;
int* p = &a;

printf(“%p %d %d”, p, *p, a);
a = 20;
printf(“%p %d %d”, p, *p, a);
```

Results:

```
00135760 14 14
00135760 20 20
```

초기화 되지 않은 변수를 사용하면 위험하다





함수–Pointer 사용 특징

- 포인터 변수도 일반 변수처럼 함수 인자, 반환형으로 사용 가능!
- 함수가 여러 개의 리턴값을 가져야 할 때 – 여러 인자들을 포인터로 받아서 수정할 수 있다!
- 실제 그 메모리 위치를 찾아가 값을 바꿀 수 있다.

- 주의!!

  함수 안에서 만들어진 local variable을 가리키는 포인터는 반환할 수 없다.

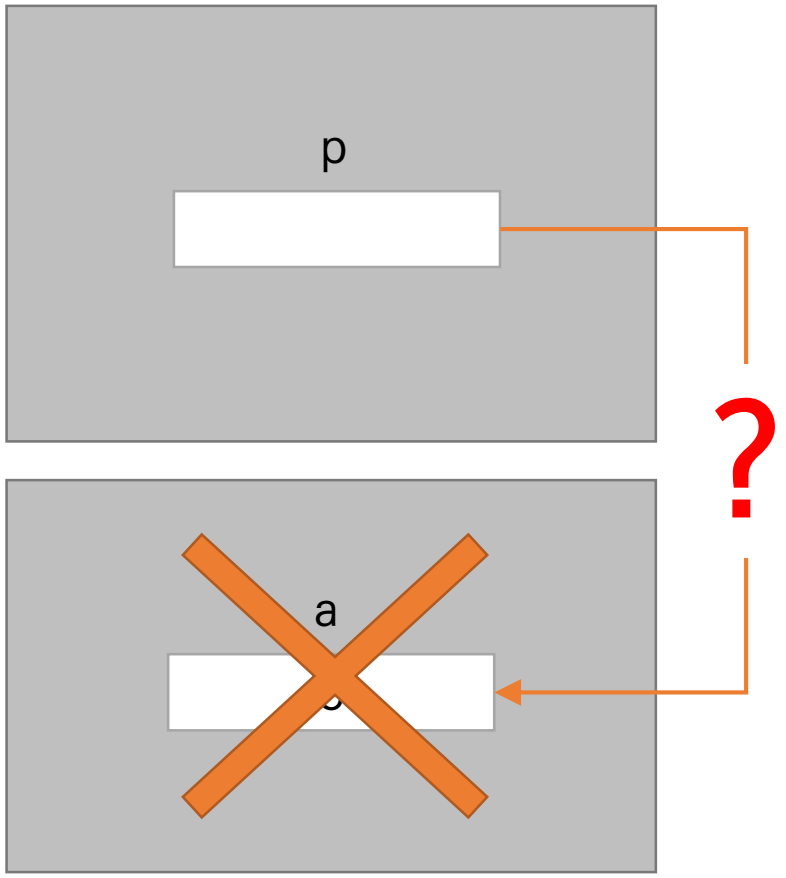
Local Variable의 주소를 리턴하는 경우

- Local variable는 그 함수(블럭)이 끝나면 사라진다.

→ 알 수 없는 곳으로의 포인터를 반환하는 것과 같다.

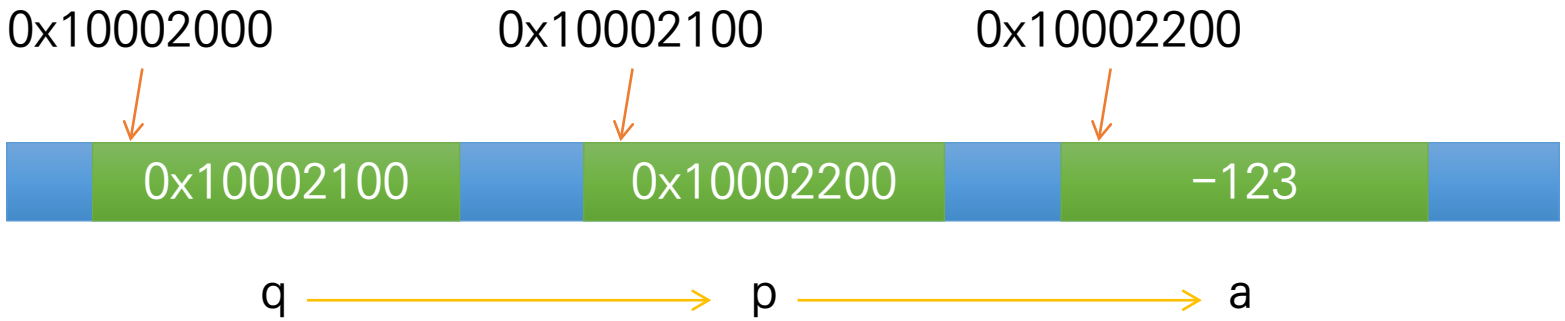
```
int main (void) {

    int* p;

    p = fun();
    printf("%d", *p);
}
```

```
int* fun (void) {

    int a = 5;

    return &a;
}
```



Pointer to Pointer (다중포인터)

- 포인터를 가리키는 포인터!
- 포인터도 주소를 저장하는 하나의 변수! 이기 때문에 메모리 특정 위치에 저장된다.




```
int a = -123;
int *p = &a;
int **q = &p;
```

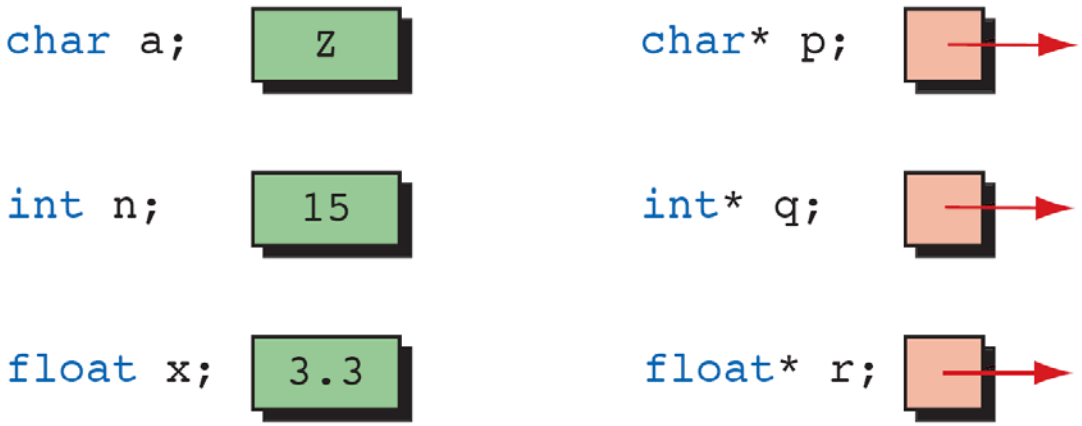
Pointer를 선언할 때 타입이 필요한 이유

int *p; char *q;

- 어차피 크기는 4byte(32 bit)로 다 똑같은 거 아닌가?
- 컴퓨터가 포인터로 메모리에 접근해 데이터를 읽어올 때!
  - 그 포인터 타입에 따라 가져오는 byte 수가 달라진다.

→“int * 포인터면 여기서 부터 4byte 까지 읽어서 정수로 쳐”

→“char * 포인터면 여기서 부터 1byte 만 읽어서 문자로 쳐”

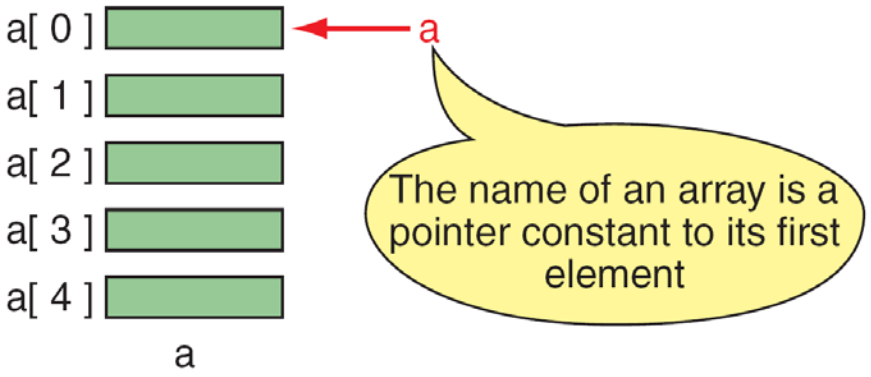


void *

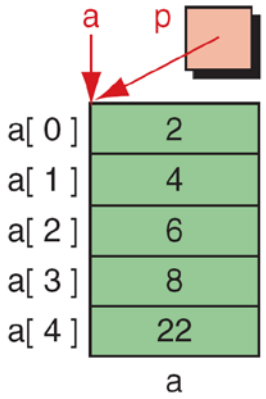
- 어떤 데이터 타입과도 연관되지 않는 일반적 형태의 포인터
  - 어떤 포인터 값도 void pointer에 넣어줄 수 있다.
  - 어떤 포인터 값에도 void pointer 값을 넣어줄 수 있다.
- 단, void pointer는 그 상태 그대로는 dereferencing 할 수 없다!
  Why? 타입이 없음 → 내가 보려는 데이터의 타입이 뭐지? → compile error
- casting을 통해서 dereferencing 할 수 있다.

```
int a = -123;
void *p;
p = &a;
printf("%d", *(int *)p);
```

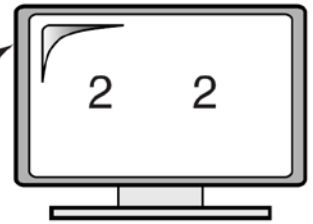
배열 포인터 (Pointer to Arrays)

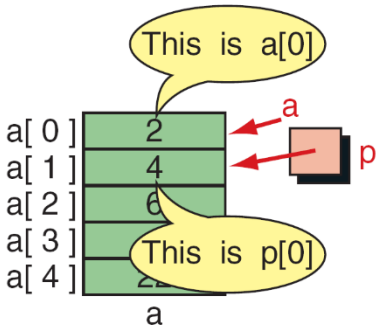


a ⟷ &a[0]

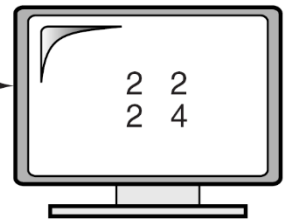



```c
#include <stdio.h>
int main (void)
{
int  a[5] = {2, 4, 6, 8, 22};
int* p    =  a;
  …
  printf("%d %d\n", a[0], *p);
  …
  return 0;
} // main
```



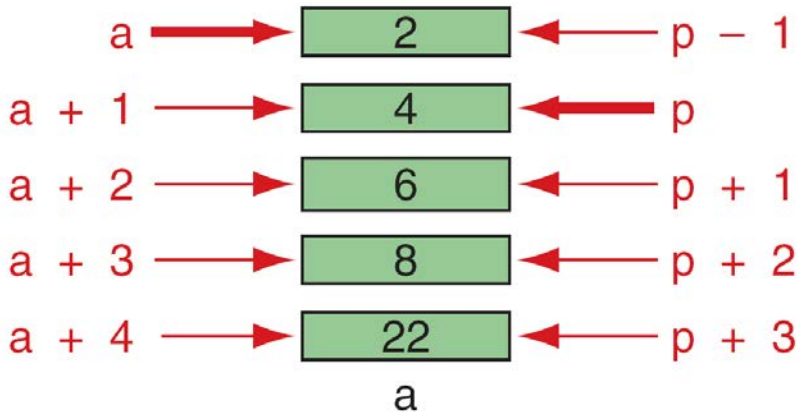


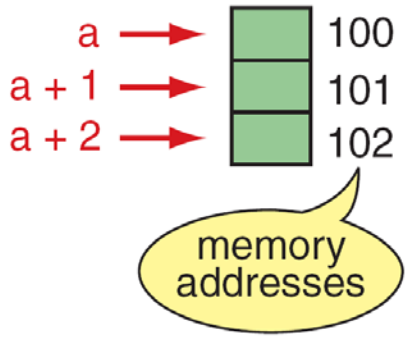

```c
#include <stdio.h>
int main (void)
{
  int  a[5] = {2, 4, 6, 8, 22};
  int* p;
  …
  p = &a[1];
  printf("%d %d", a[0], p[-1]);
  printf("\n");
  printf("%d %d", a[1], p[0]);
  …
} // main
```



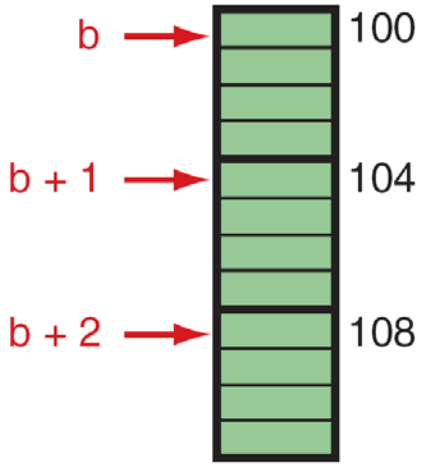
포인터 연산

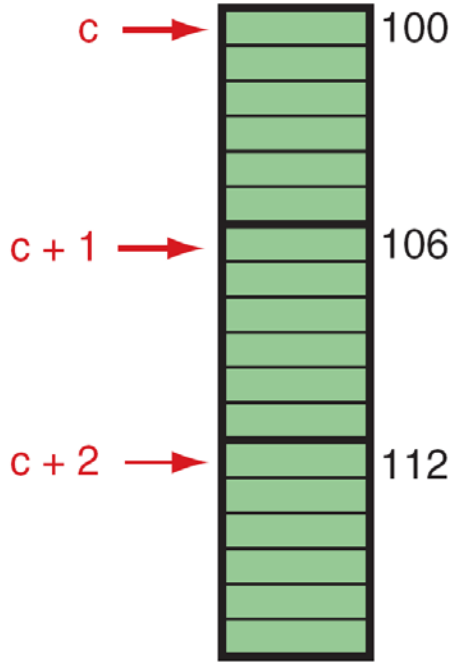
- 포인터 p → p ± n → p + n * (sizeof (one element))



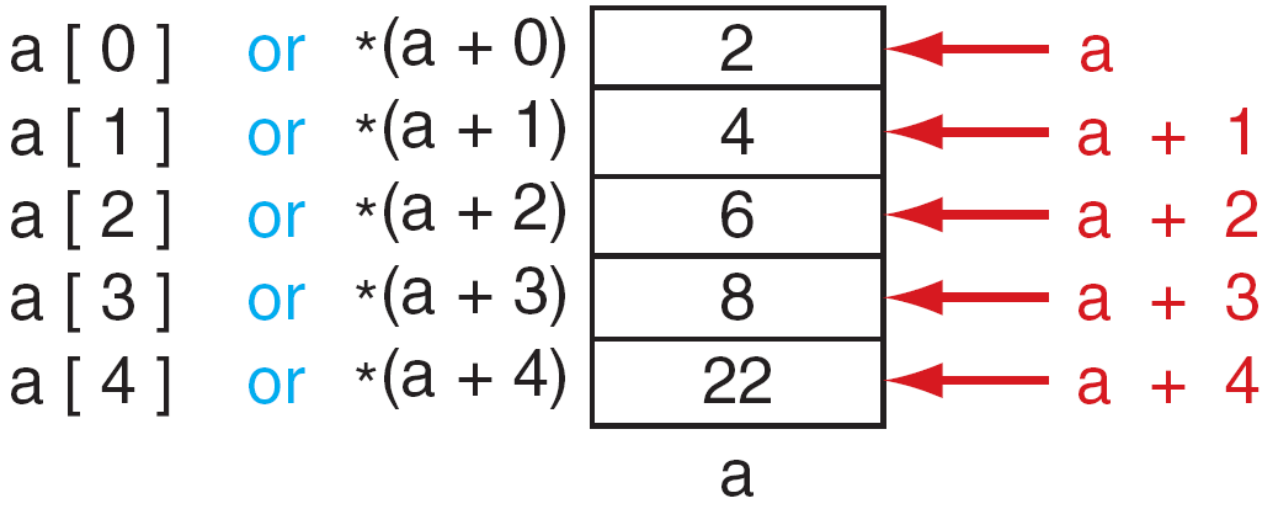



```
char  a[3];
int   b[3];
float c[3];
```





Dereferencing



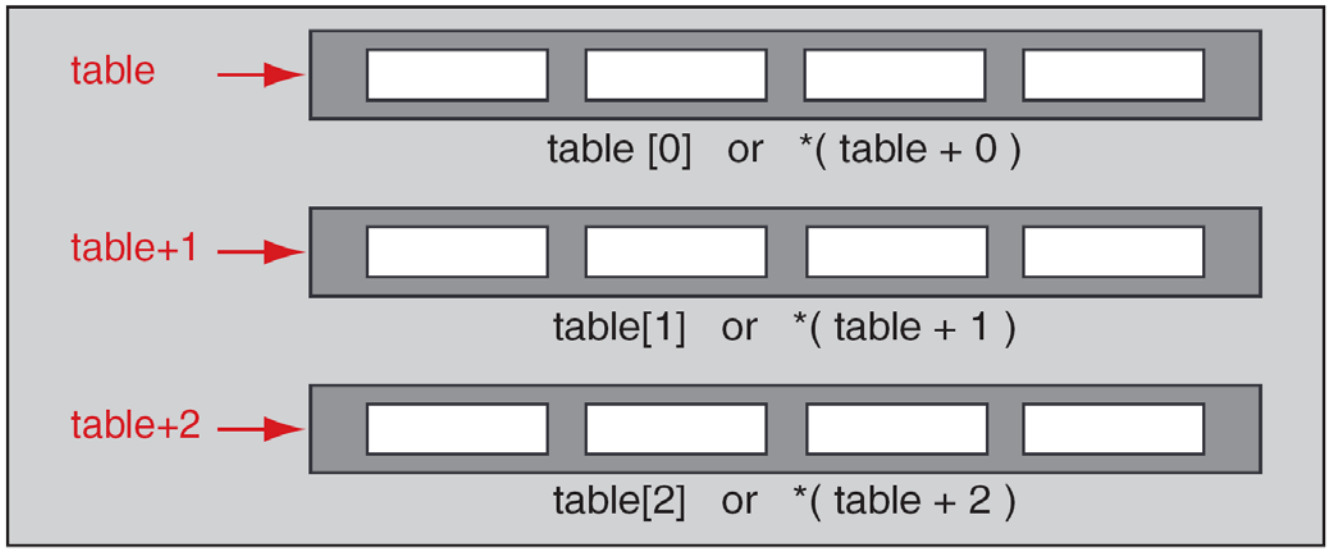
*(a + i) ⟺ a[i]

2차원 Array

table[2] == *(table+2)

table[2][1]
== (*(table+2))[1]
==*(table[2]+1)
==*(*(table+2)+1)

- 헷갈리니 다차원 배열에서는 Index를 사용하자



int table[3][4] ;

```
for (i = 0; i < 3; i++)
  {
   for (j = 0; j < 4; j++)
      printf("%6d", *(*(table + i) + j));
   printf( "\n" );
  } // for i
```

Print Table

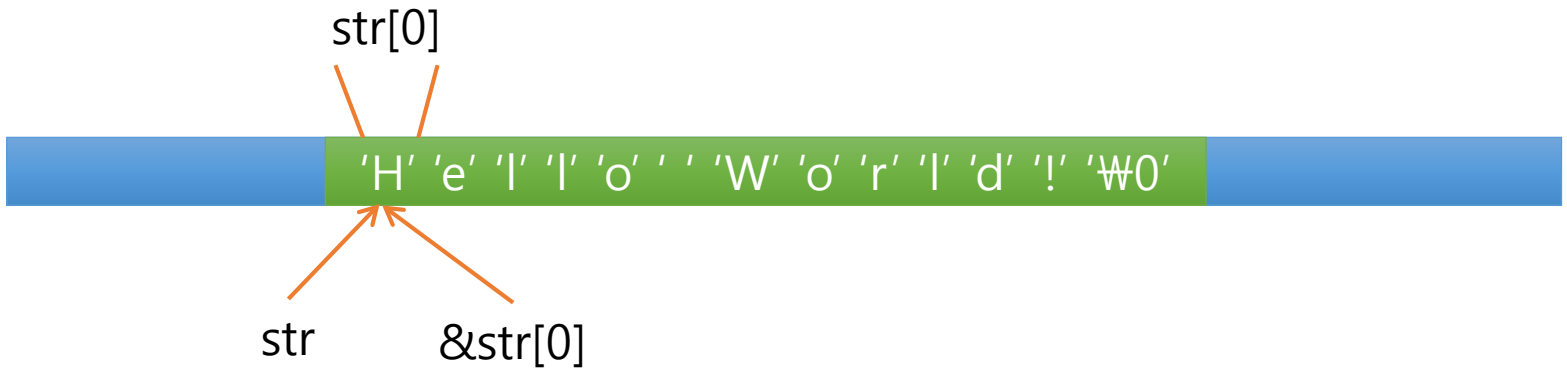
Arithmetic Operations on Pointers

- p + 5, 5 + p, p − 5
- p1 − p2
- p++, −−p
- p1 〉= p2
- p1 != p2

Long Form	Short Form
if (ptr == NULL)	if (!ptr)
if (ptr != NULL)	if (ptr)

문자열과 포인터

- 다시한번! 여러개의 문자+'\0' 이 들어있는 배열
- 배열의 이름은 첫번째 element의 주소와 같다.



- 그래서 scanf로 %s 받을 때 & 안 붙인다!

문자열 한 줄 다 입력받기

char buf[10];

- gets(char *buf)
  - gets(buf); //stdin으로 부터 입력 받는다.
    //길이 제한 없음 (보안상 취약)
    //newline 이나 EOF 만날 때 까지
    //끝에 '\0'을 자동으로 붙여준다.
- fgets(char *buf, unsigned size, FILE *file)
  - Fgets(buf, 10, stdin); //지정 stream으로 부터 입력 받는다.
    //최대 크기를 지정해 줄 수있다.(널 포함)
    //newline 이나 EOF 만날 때 까지
    //끝에 '\0'을 자동으로 붙여준다.
    //문자는 최대 size-1 개 까지 받는다.

배열 함수에 넘겨주기

- 1차원 배열
  Int ary1[10];
  void function(int *ary);
  void function(int ary[]);
- 2차원 배열
  int ary2[5][6];
  void function(int (*ary)[6]);
  void function(int ary[][6]);
- 3차원 배열
  (int ary[][3][4])

배열 받기 연습

- Int* arr1[3] ; –〉 (int** ptr1)
- Int * arr2[3][5]; –〉 (int* (*ptr2)[5])
- Int** arr3[5]; –〉 ?
- Int *** arr4[3][5]; –〉 ?

오늘 할 것

- Dynamic Memory Allocation

Dynamic Memory Allocation

동적할당

왜 쓰나요?

- 배열의 크기를 입력 받아 그 크기만큼의 배열을 만들고 싶을 때

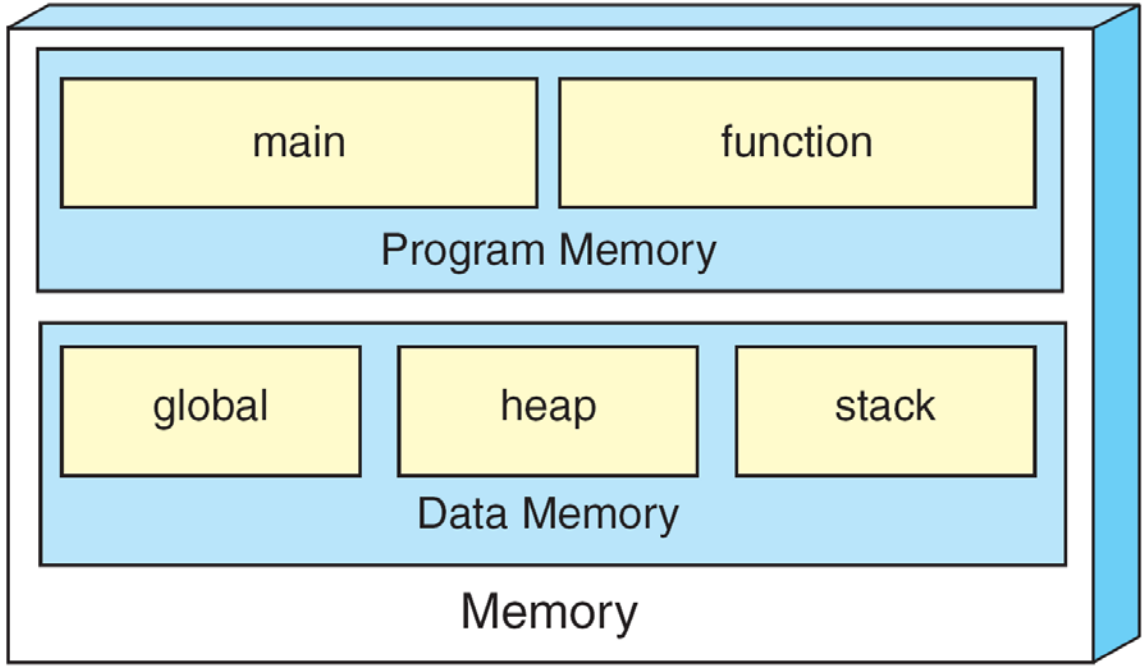
```
int main(void) {
    int size;

    scanf(“%d”, &size);

    int array[size];
}
```

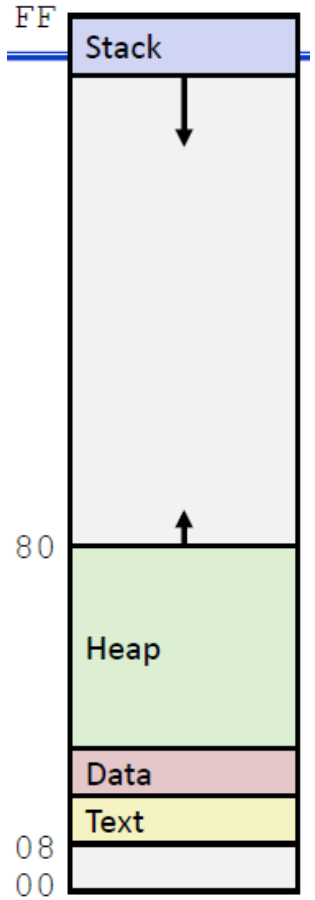
Compile Error! 선언은 항상 맨 위에 와야 한다.
선언 후에 배열이 할당되어야 하는데… 이럴 때는 어떻게?

- 프로그램 실행 중에 메모리를 할당해 데이터를 저장할 공간을 생성하는 방법

A Conceptual View of Memory

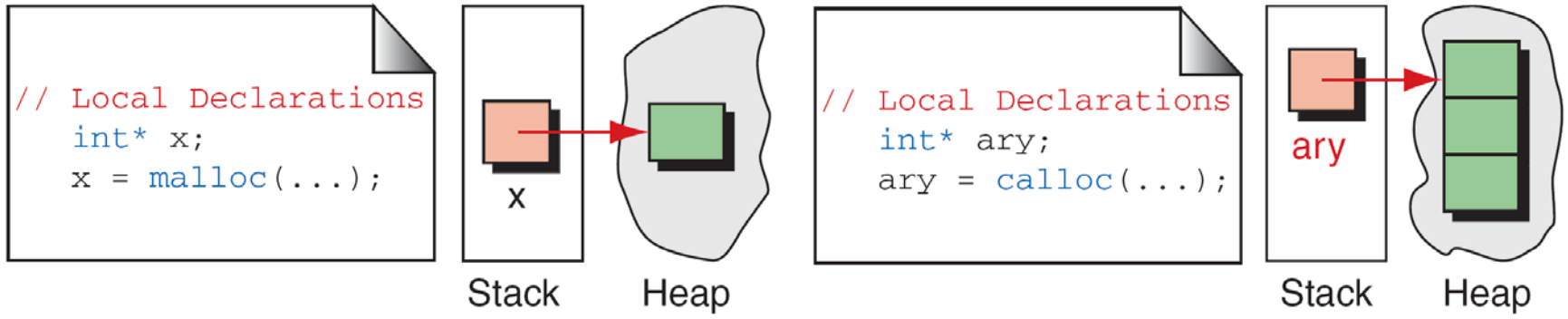
- We can refer to memory allocated in the heap only through a pointer.





Accessing Dynamic Memory

(a) Static Memory Allocation



(b) Dynamic Memory Allocation

Dynamic Memory Allocation

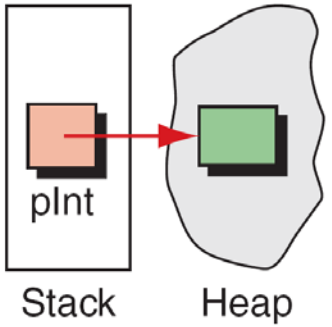
- ⟨stdlib.h⟩

- void *malloc (size_t size);
- void *calloc (size_t element-count, size_t element-size);
- void *realloc (void* ptr, size_t newSize);
- void free (void* ptr);

malloc

- 지정하는 크기(byte) 만큼 메모리(힙)를 할당
- 기본적으로 void *로 리턴 → 원하는 타입으로 Cating 해줘야 한다
- 할당에 실패한 경우 NULL 리턴
- void *malloc (size_t size);

```
if (!(pInt = malloc(sizeof(int))))
    // No memory available
    exit (100) ;
// Memory available
…
```



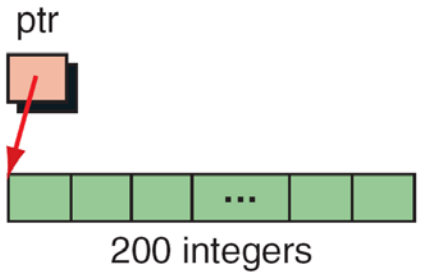
- int* p = (int *)malloc(sizeof(int) * 4);
  - int형의 크기가 4byte이기 때문에 16byte의 공간이 동적으로 할당
- char *str = (char *)malloc(sizeof(char) * 10);

calloc

- void *calloc (size_t element_count, size_t element_size);

- int *p = (int *)calloc(4, sizeof(int));
- char *str = (char *)calloc(10, sizeof(char));

malloc(sizeof(int)*4) ↔ calloc(4, sizeof(int))




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

// Memory available
…
```

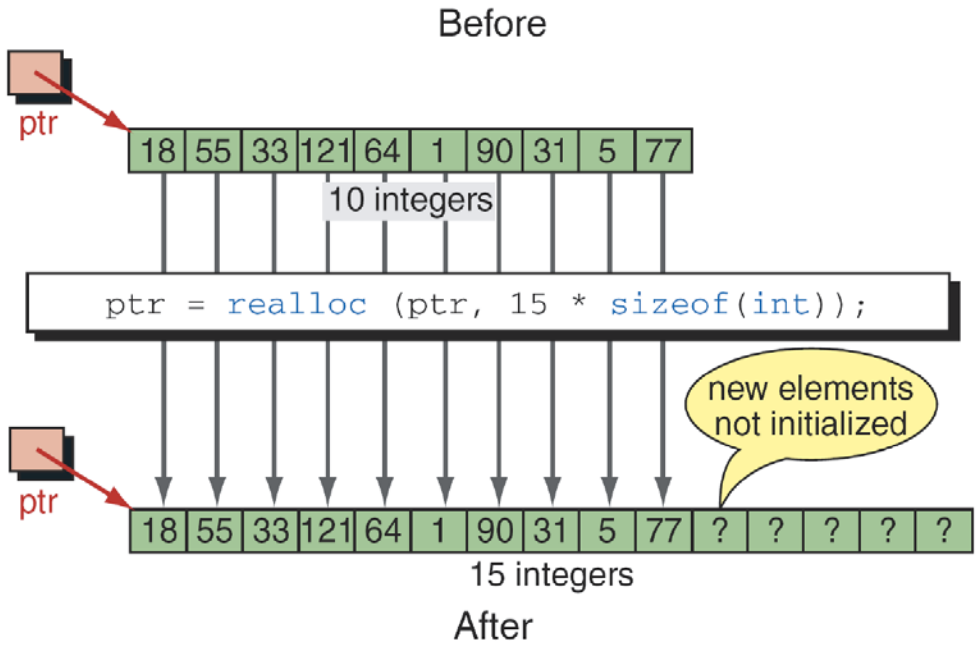
realloc

- 할당받은 메모리 공간을 새로운 크기로 재할당

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

- int *p = (int *)calloc(4, sizeof(int));
  p = (int *)realloc(p, sizof(int) * 6);

- char *str = (char *)calloc(10, sizeof(char));
  str = (char *)realloc(str, 20*sizeof(char));

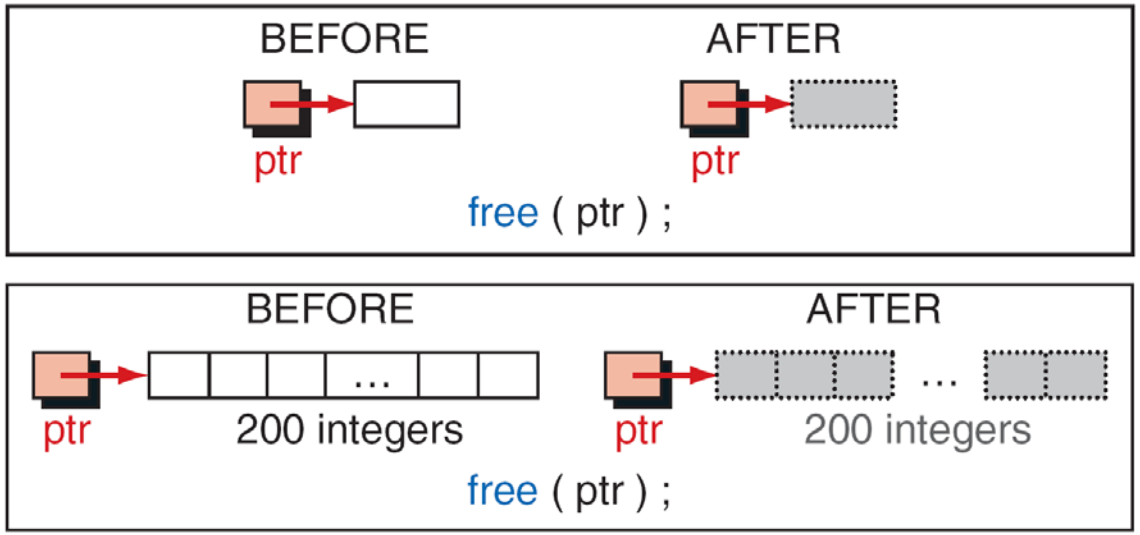


free

- 동적 할당한 메모리 반환
- 사용하지 않을 때 & 프로그램의 마지막에 반드시 반환해야 한다!

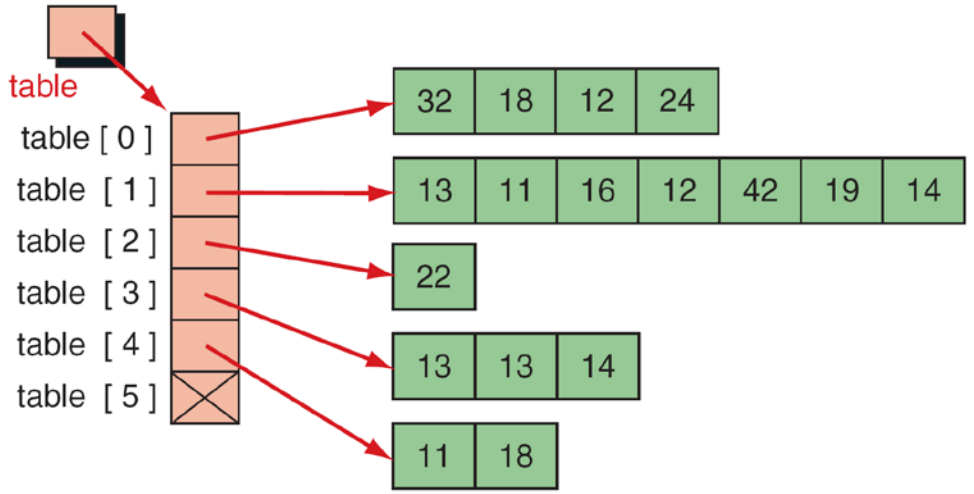
- void free (void * ptr);

- free(p);
- free(str);



동적할당으로 2차원 배열 만들기 – 포인터 배열

```
int **table;
table = (int **)calloc (3, sizeof(int *));
table[0] = (int *)calloc(4, sizeof(int));
...
...
...
```




```
table = (int**)calloc (rowNum + 1, sizeof(int*));

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

다음시간

- Enumeration, Structure, Union
- String (Advanced)