

전산 SMP 10주차

2014. 12. 03

김범수

bskim45@gmail.com

Special thanks to 박기석 (kisuk0521@gmail.com)

지난 내용 복습

Dynamic Memory Allocation

왜 쓰나요?

- 배열의 크기를 입력 받아 그 크기만큼의 배열을 만들고 싶을 때

```
int main(void) {
```

```
    int size;
```

```
    scanf("%d", &size);
```

```
    int array[size];
```

```
}
```



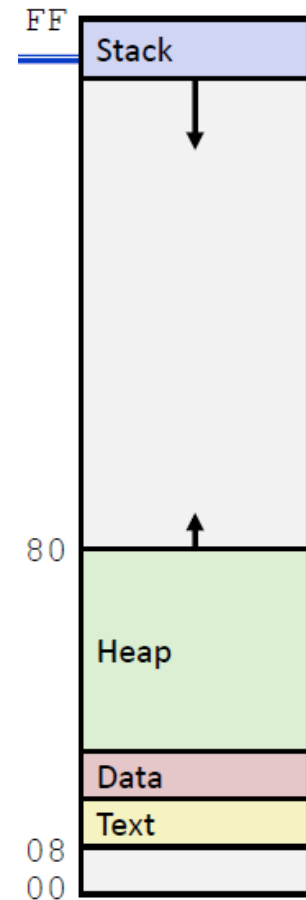
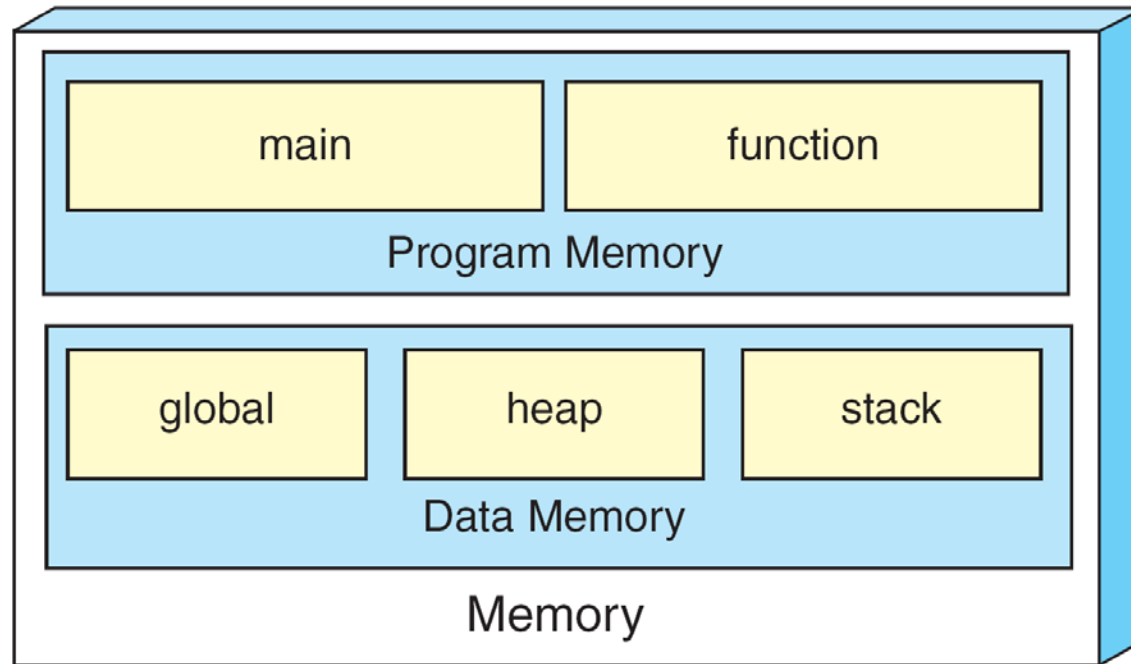
Compile Error! 선언은 항상 맨 위에 와야 한다.

선언 후에 배열이 할당되어야 하는데... 이럴 때는 어떻게?

- 프로그램 실행 중에 메모리를 할당해 데이터를 저장할 공간을 생성하는 방법

A Conceptual View of Memory

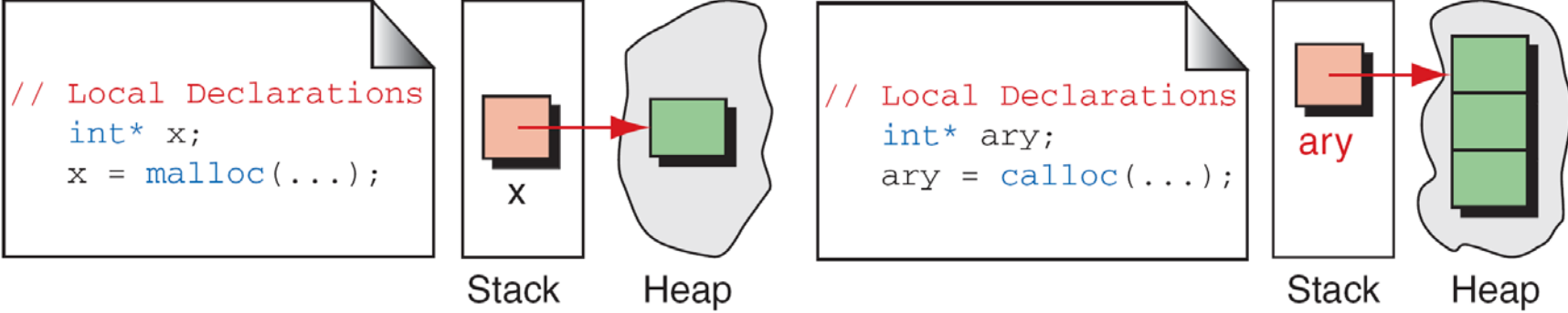
- We can refer to memory allocated in the heap only through a pointer.



Accessing Dynamic Memory



(a) Static Memory Allocation



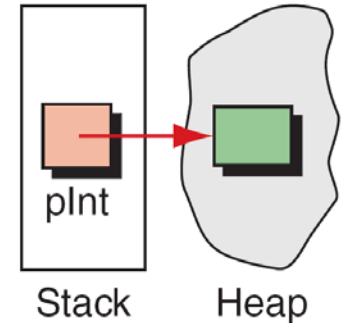
(b) Dynamic Memory Allocation

malloc

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

- void *malloc (size_t size);

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

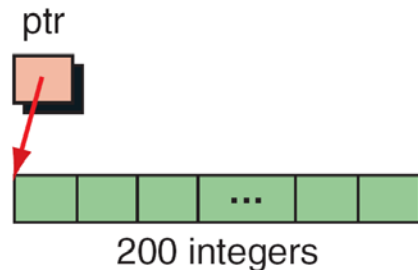


- int* p = (int *)malloc(sizeof(int) * 4);
 - int형의 크기가 4byte이기 때문에 16byte의 공간이 동적으로 할당
- char *str = (char *)malloc(sizeof(char) * 10);

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));`

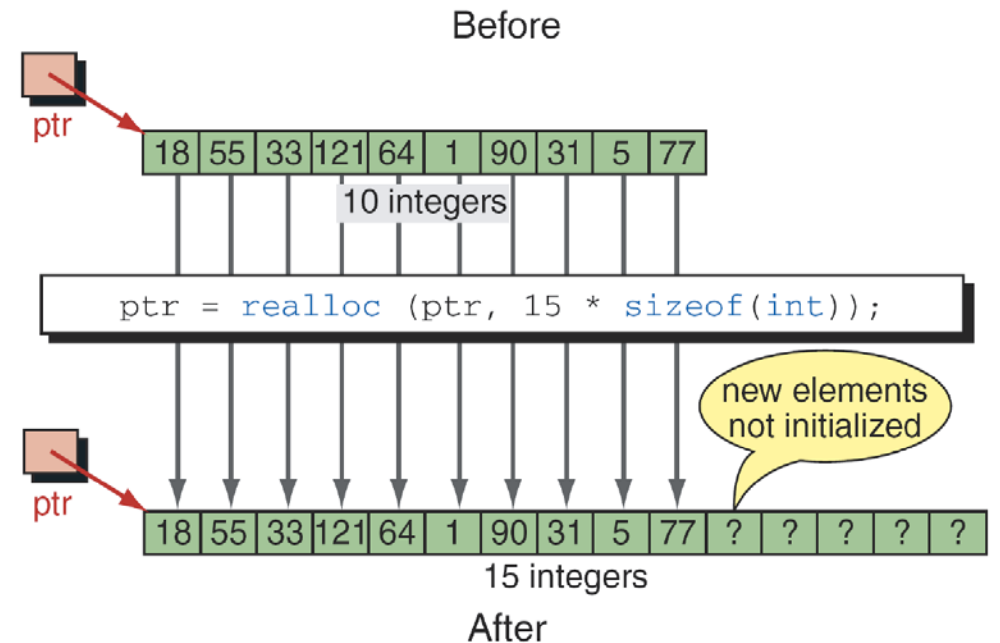
`malloc(sizeof(int)*4) ↔ calloc(4, sizeof(int))`



```
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, sizeof(int) * 6);`
- `char *str = (char *)calloc(10, sizeof(char));`
`str = (char *)realloc(str, 20*sizeof(char));`



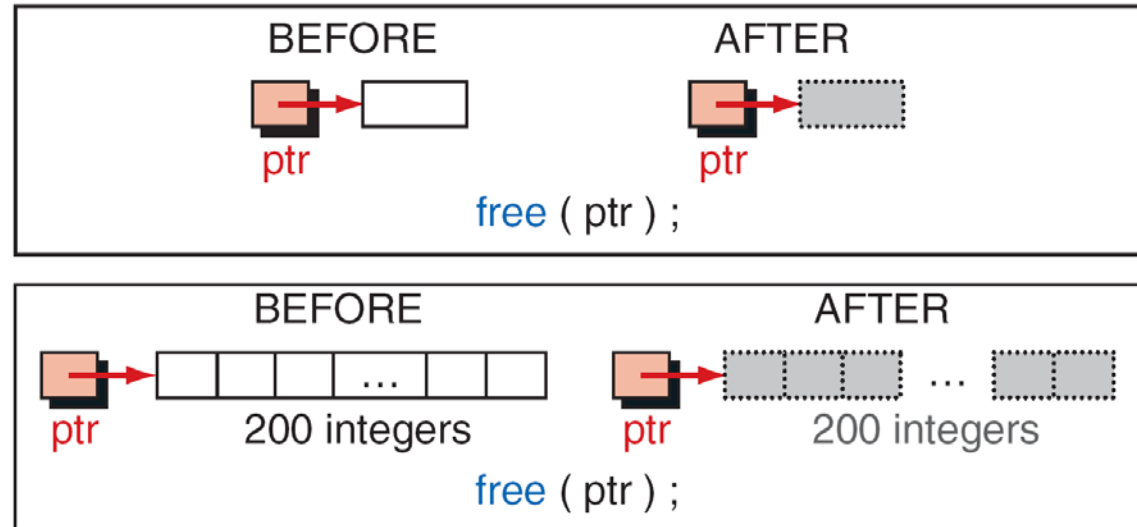
free

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

- `void free (void * ptr);`

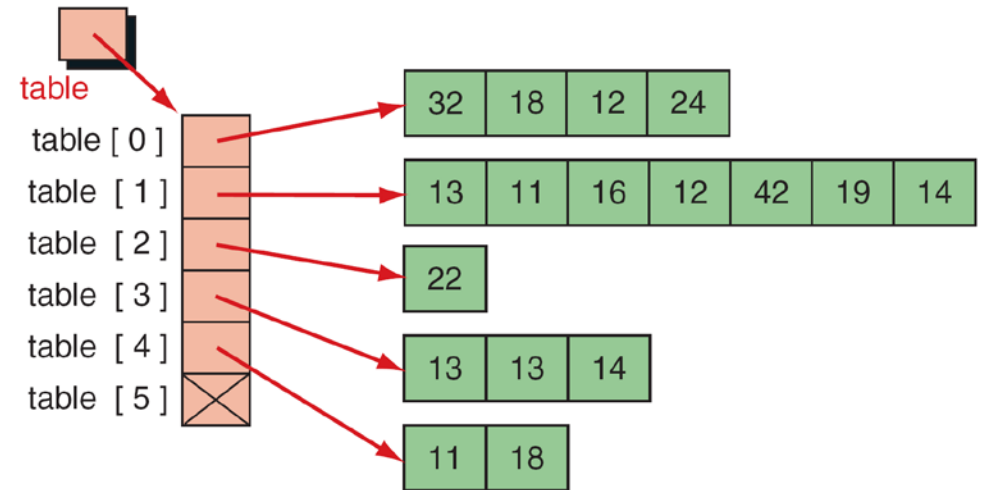
- `free(p);`

- `free(str);`



동적할당으로 2차원 배열 – 포인터 배열

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

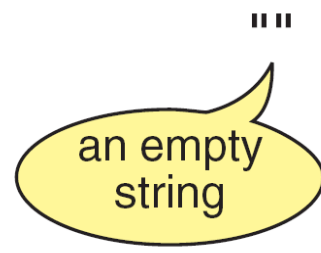
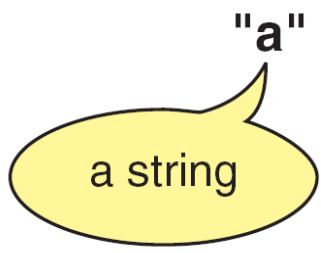
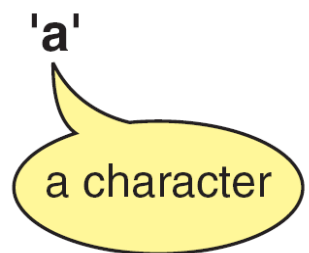
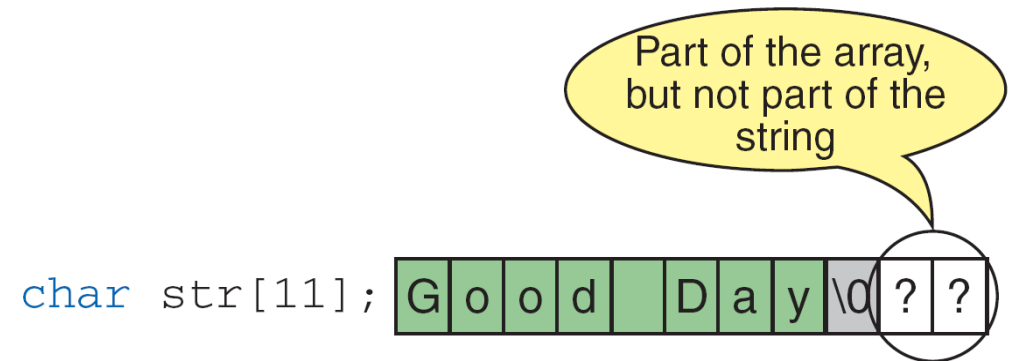
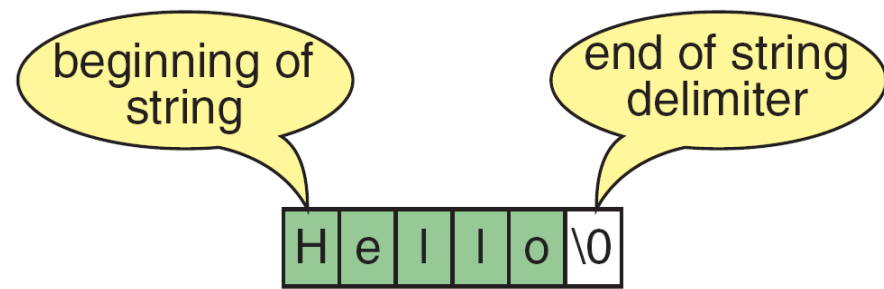
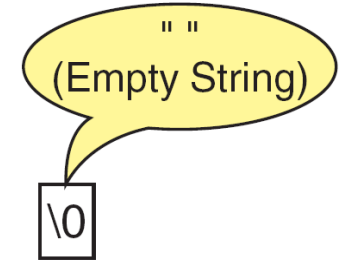
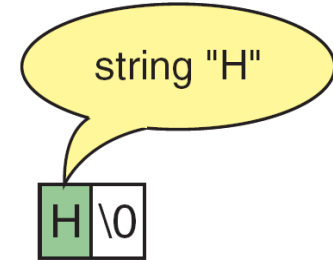
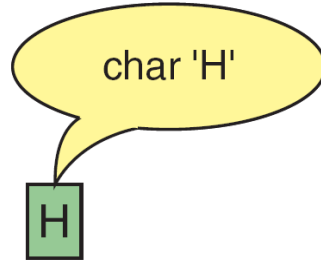


```
table = (int**)calloc (rowNum + 1, sizeof(int*));  
table[0] = (int*)calloc (4, sizeof(int));  
table[1] = (int*)calloc (7, sizeof(int));  
table[2] = (int*)calloc (1, sizeof(int));  
table[3] = (int*)calloc (3, sizeof(int));  
table[4] = (int*)calloc (2, sizeof(int));  
table[5] = NULL;
```

String

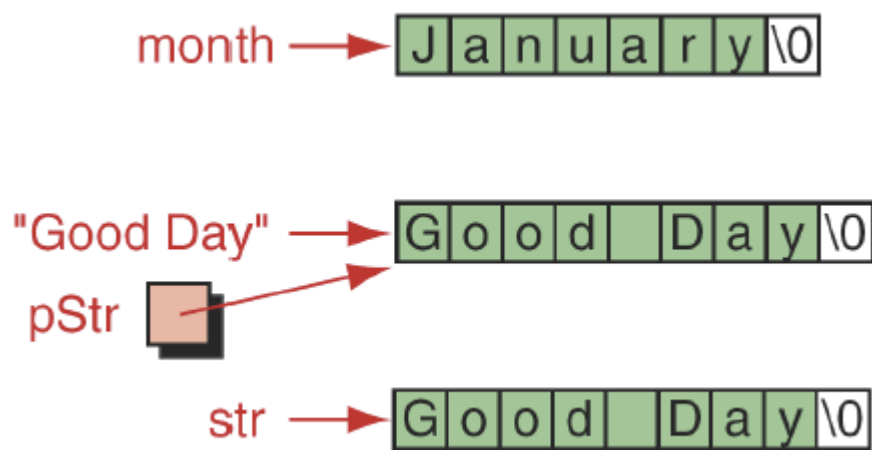
문자열(string)도 결국은 배열이다

- char 형의 배열
- 끝에 널문자 '\0'



문자열의 선언

- `char str[15] = "Good Day";`
 - 할당된 용량을 모두 사용할 필요는 없습니다.
- `char month[] = "January";`
 - 배열크기가 문자열의 길이에 맞춰 자동으로 설정
- `char *pStr = "Good Day";`
- `char str[9] = {'G', 'o', 'o', 'd', ' ', 'D', 'a', 'y', '\0'};`



```
// Local Declarations  
char str[9];
```

(a) String Declaration

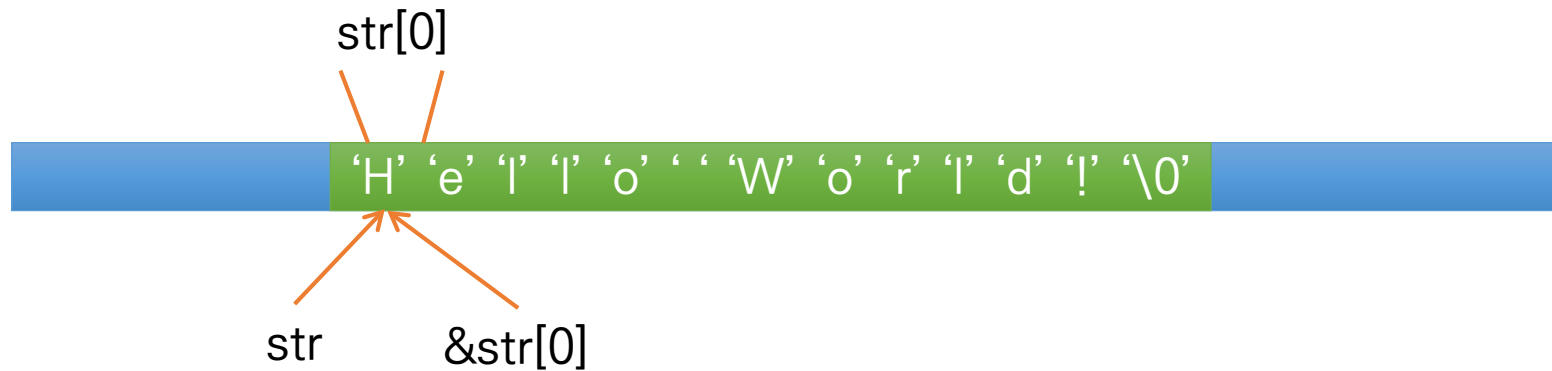
```
// Local Declarations  
char* pStr;
```

(b) String Pointer Declaration



문자열의 포인터

- 배열의 이름은 첫번째 element의 주소와 같다.
- 그래서 scanf로 %s 받을 때 & 안 붙인다!



문자열 포인터 사용 주의점

```
// Local Declarations  
char str[9];
```

(a) String Declaration



```
// Local Declarations  
char* pStr;
```

(b) String Pointer Declaration



- 반드시 동적 할당 해주거나 어떤 문자열의 주소를 넣어주어야 한다.

String Copy?

- `char str1[6] = "Hello";`
`char str2[6];`
`str2 = str1; // ?`

String Input/Output Functions

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

formatted input/output functions

- scanf/fscanf
- printf/fprintf

special set of string-only functions

- get string (gets/fgets)
- put string (puts/fputs).

FLUSH

- 스트림 버퍼를 비우는 역할을 한다.
- Scanf 등 입력받는 함수 사용 시
→ The string conversion code(s) skips whitespace.

Test)

```
int a; char b;  
scanf("%d", &a);  
scanf("%c",&b);  
printf("%d %c\n", a, b);
```

```
1 { // Read Month  
2   #define FLUSH while (getchar() != '\n')  
3   char month[10];  
4  
5   printf("Please enter a month. ");  
6   scanf("%9s", month);  
7   FLUSH;  
8 }
```

```
fflush();
```

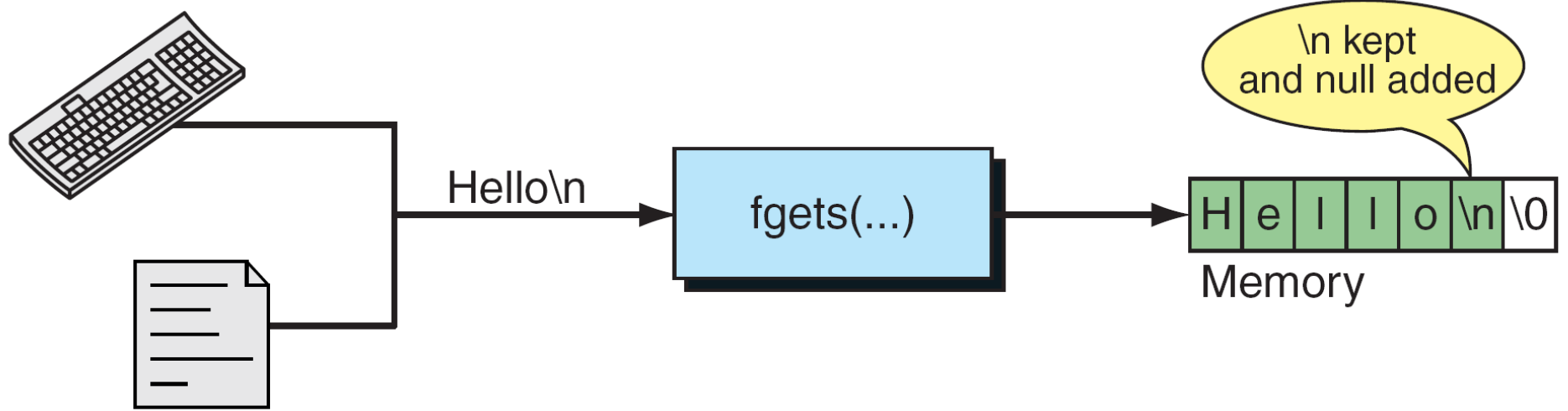
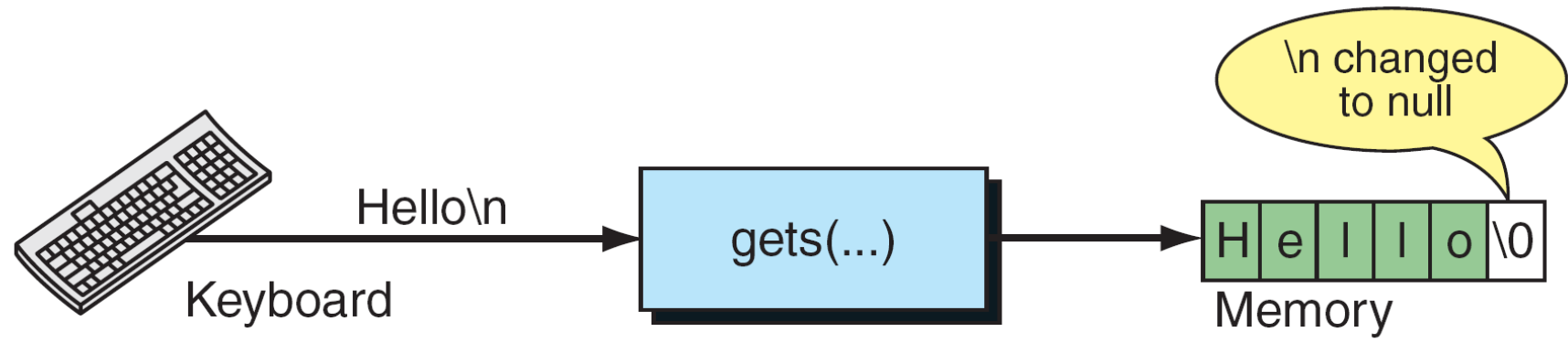
```
#define FLUSH while(getchar != '\n')
```

Formatted String Input

- `char str[10];`
- `scanf("%9s", str);`
- `str` - 배열 포인터 → `&`가 붙지 않음

- 입력 문자 개수를 반드시 정의해 주자

String-only Input: gets/fgets



gets(...)

```
char * gets(char *str);
```

Reads characters from stdin, until either a newline or EOF

Params

- str: pointer to an array of chars
→ 길이 제한 없음 (보안상 취약)

Return value

- Success: returns str (incl. '\0' at the end)
- EOF: feof() set, 여태까지 읽은 str return
- 아무것도 못 읽으면: NULL

fgets(...)

```
char * fgets (char *str, int num, FILE *stream);
```

Reads characters from stream, until (num-1) characters have been read OR either a newline or the EOF

Params

- str : pointer to an array of chars
- num : Maximum number of characters (incl. null) → 문자는 최대 (num-1)
- stream : Pointer to a FILE that identifies an input stream

Return value

- Success: returns str (incl. '\0' at the end)
- EOF: sets EOF(feof), 여태까지 읽은 str 리턴
- 아무것도 못 읽으면: NULL

Difference btw. Input functions

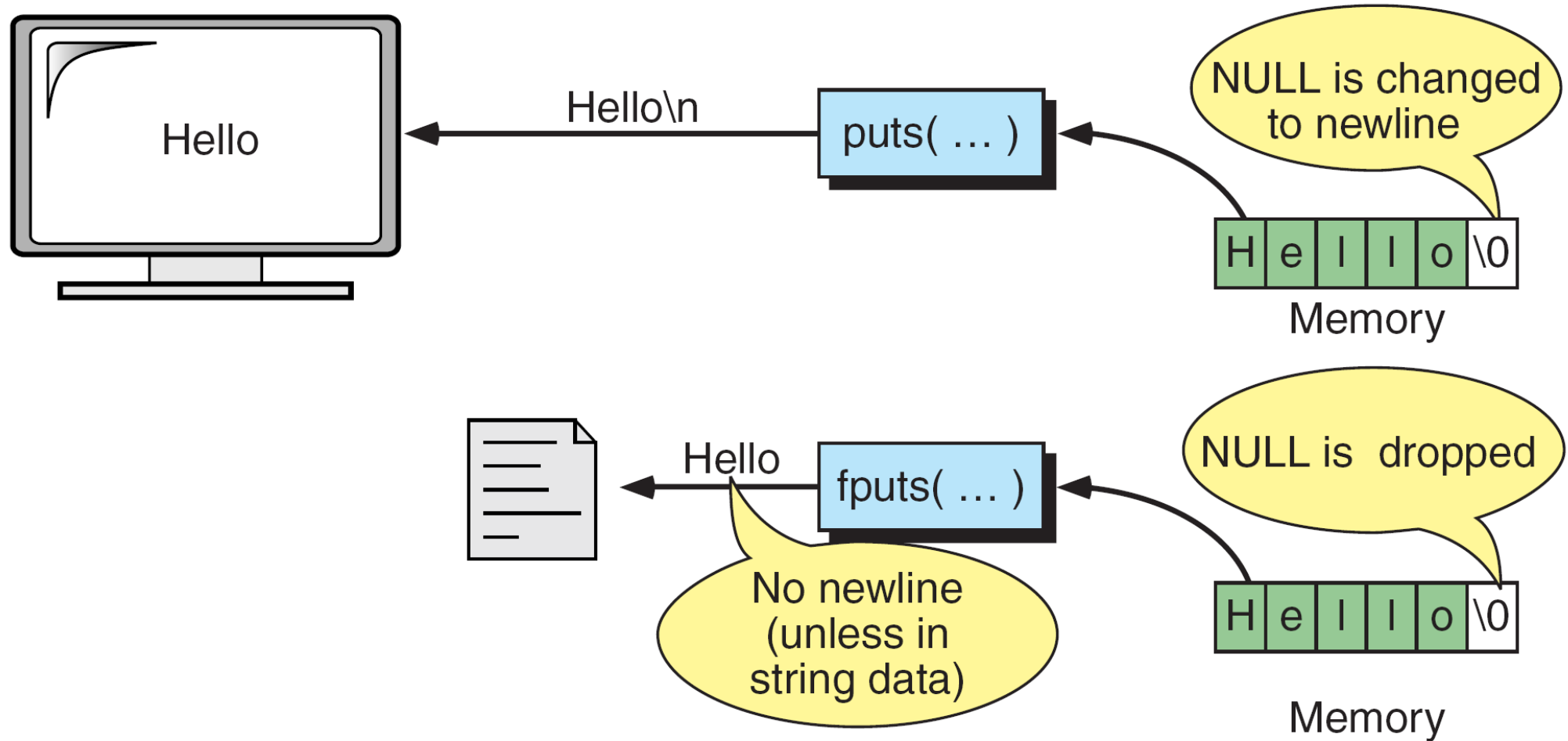
gets/fgets

- gets does not include newline (“\n”), fgets does.
- gets does not allow to specify a maximum size for *str* (which can lead to buffer overflows).

Scanf/gets

- scanf는 단어 단위로(space), gets는 줄 단위로(newline)

String-only Output: puts/fputs



puts(...)

```
int puts ( const char * str );
```

Writes str to stdout and appends a newline character ('\n') until null('0\').
Terminating null-character is not copied to the stream.

Params

- str: pointer to an array of chars

Return value

- Success: returns a non-negative value
- Error: returns EOF(-1) and sets the error indicator (ferror)

fputs(...)

```
int fputs ( const char * str, FILE * stream );
```

Writes *str* to the stream until null ('\0'). Terminating null-character is not copied to the stream.

Params

- *str*: pointer to an array of chars
- *stream*: Pointer to a FILE object that identifies an output stream.

Return value

- Success: returns a non-negative value
- Error: returns EOF(-1) and sets the error indicator (ferror)

Difference btw. Output functions

- puts appends a newline('\n) at the end automatically
- fputs does not write additional characters

String Manipulation Functions

- `#include <string.h>`
- String Length and String Copy
- String Compare and String Concatenate
- Character in String
- Search for a Substring and Search for Character in Set
- String Span and String Token
- String to Number

Assn #4 – Prob 1: String 처리 함수 구현하기

- `int mystrlen(char *str);`
- `char *mystrcpy(char *toStr, char *fromStr);`
- `char *mystrcmp(char *str1, char *str2);`
- `char *mystrcat(char *str1, char *str2);`
- `char *mystrchr(char *str, char ch);`
- `int myatoi(char *str);` → 이걸 저번 어싸인에서 했고

- 위에서 기본적인 원리는 다 했죠? ^^

Assn #4 – Prob 2

- 최소 5개 이상의 함수 작성 - 파일로부터 리스트를 생성하는 부분(1개의 함수)과 각 메뉴(4개의 함수)
- 특정 학생의 정보 및 과목 정보는 구조체의 변수에 저장, 프로그램 내부에서 동적 할당된 구조체 포인터의 배열을 이용하여 관리 = 필요할 때 마다 동적 할당 해서 쓰세요
- `C:\> assn4.exe students.txt`
→ Argument Passing - `int main(int argc, char* args[])`
- 예외 처리 - 파일 읽기에서 읽을 파일이 존재하지 않을 경우: `fopen`의 “r” 모드 리턴 값 `NULL`
- 파일 읽기 - `fgets`가 `NULL`을 리턴할 때까지(EOF)

Assn #4 – Prob 2

```
typedef struct {  
    int id;  
    char name[11];  
    char dept[7];  
    int level;  
    int sub_num;  
    TAKING_SUBJECTS  
    subjects[MAX_TAKING_SUBJECT_NUM];  
} STUDENT;
```

```
STUDENT* stu_list[MAX_STUDENT_NUM];  
SUBJECT* sub_list[MAX_SUBJECT_NUM];
```

1. 학생 별 성적

stu_list 한번 돌면서 STUDENT 안에 subjects 배열 한번 돌고

2. 과목 별 성적

학생 명단을 출력하는 게 좀 짜증나는데, 역시 stu_list 돌면서 subjects 배열에 해당 과목이 있으면 출력해 주면 되겠죠~

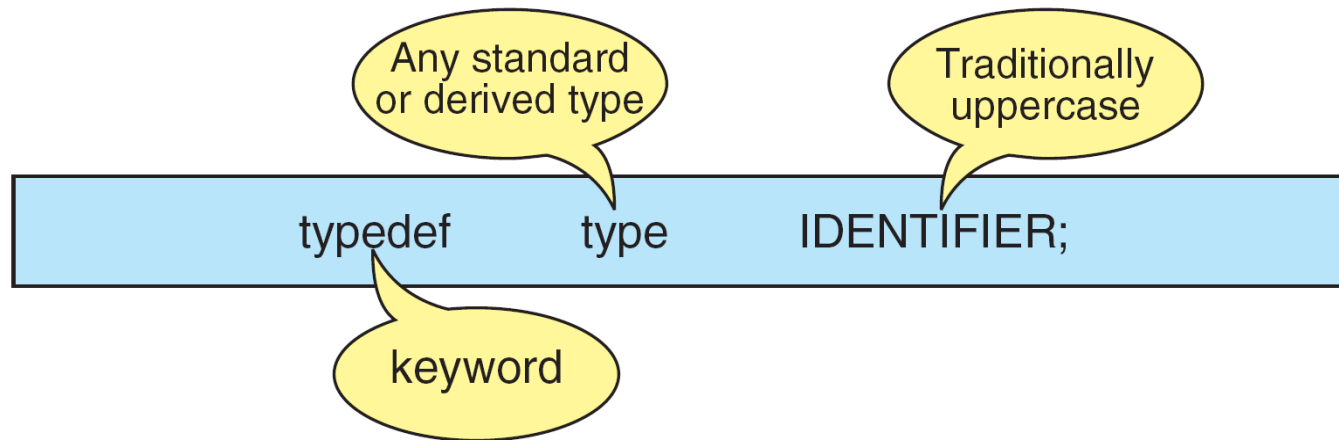
3. 저장하기

For 문으로 배열 돌면서 주어진 틀에 맞춰서 fputs 만 잘 써주면 되겠네요~

Structure

The Type Definition (typedef)

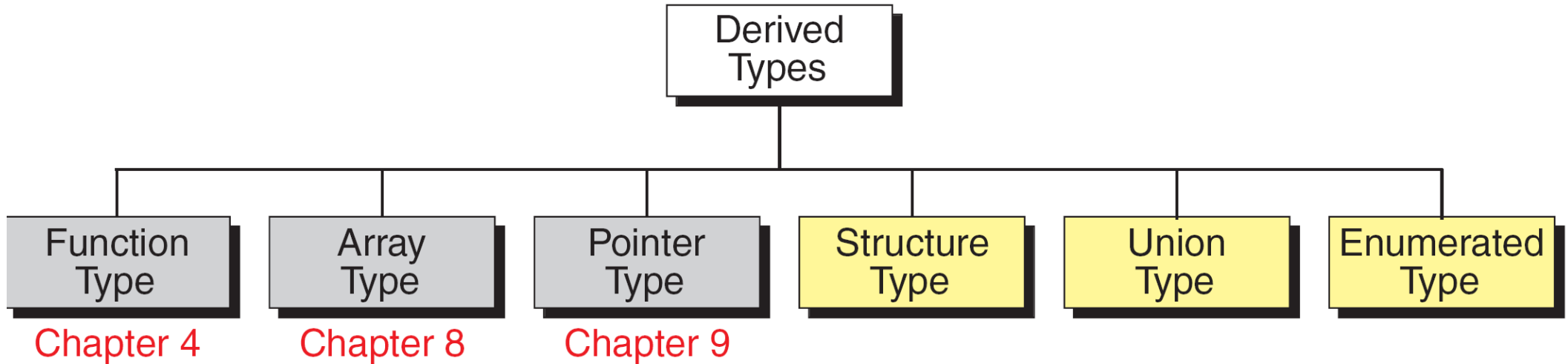
- A type definition, typedef, gives a name to a data type by creating a new type that can then be used anywhere a type is permitted.
- 프로그래머는 길게 쓰는 것을 싫어한다



```
typedef int INGETER;
```

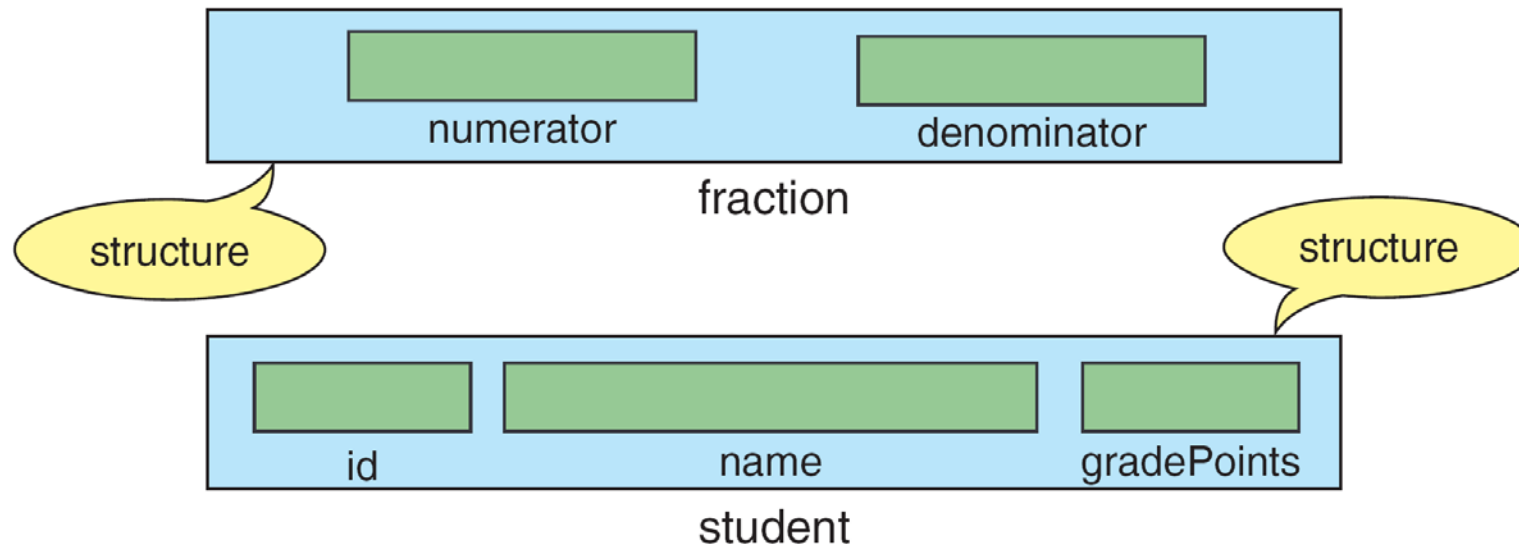
Derived Type

- 프로그래머가 임의로 만든 타입



Structure Type

- 관련되는 element 들의 집합.
- 어떤 타입의 변수라도 올 수 있다.
- 단, 함수는 구조체의 element로 안 된다. 오직 변수만 가능



Structure Declaration

```
// Global Type Declarations
```

```
struct STUDENT  
{  
    char id[10];  
    char name[26];  
    int gradePts;  
} ;
```

```
// Local Declarations
```

```
struct STUDENT aStudent;
```

```
// Global Type Declarations
```

```
typedef struct  
{  
    char id[10];  
    char name[26];  
    int gradePts;  
} STUDENT;
```

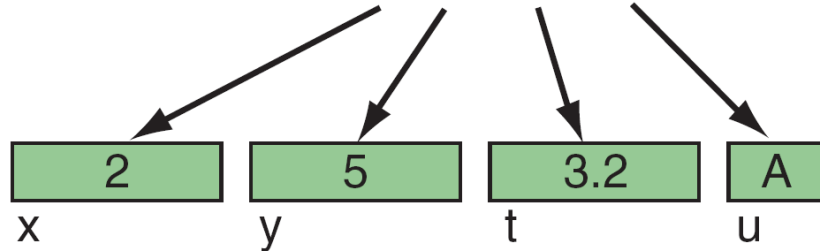
```
// Local Declarations
```

```
STUDENT aStudent;
```

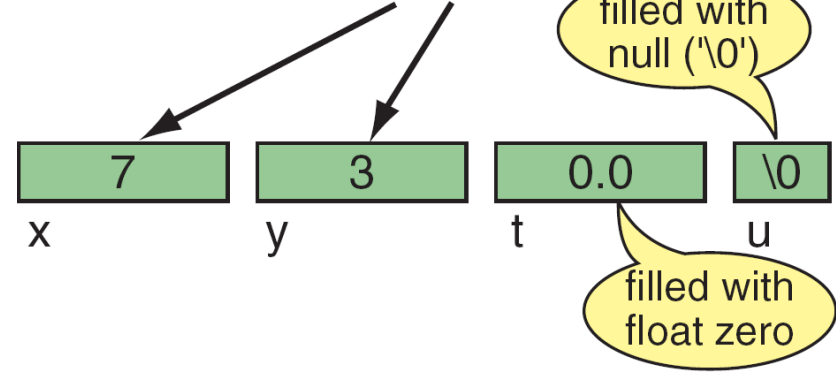
Initializing Structures

```
typedef struct
{
    int    x;
    int    y;
    float  t;
    char   u;
} SAMPLE;
```

SAMPLE sam1 = { 2, 5, 3.2, 'A' };

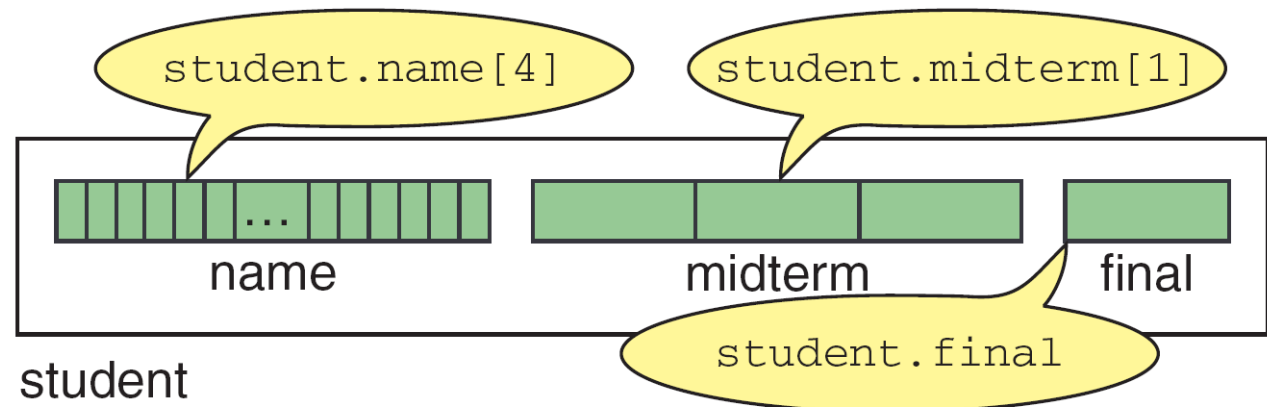


SAMPLE sam2 = { 7, 3 };

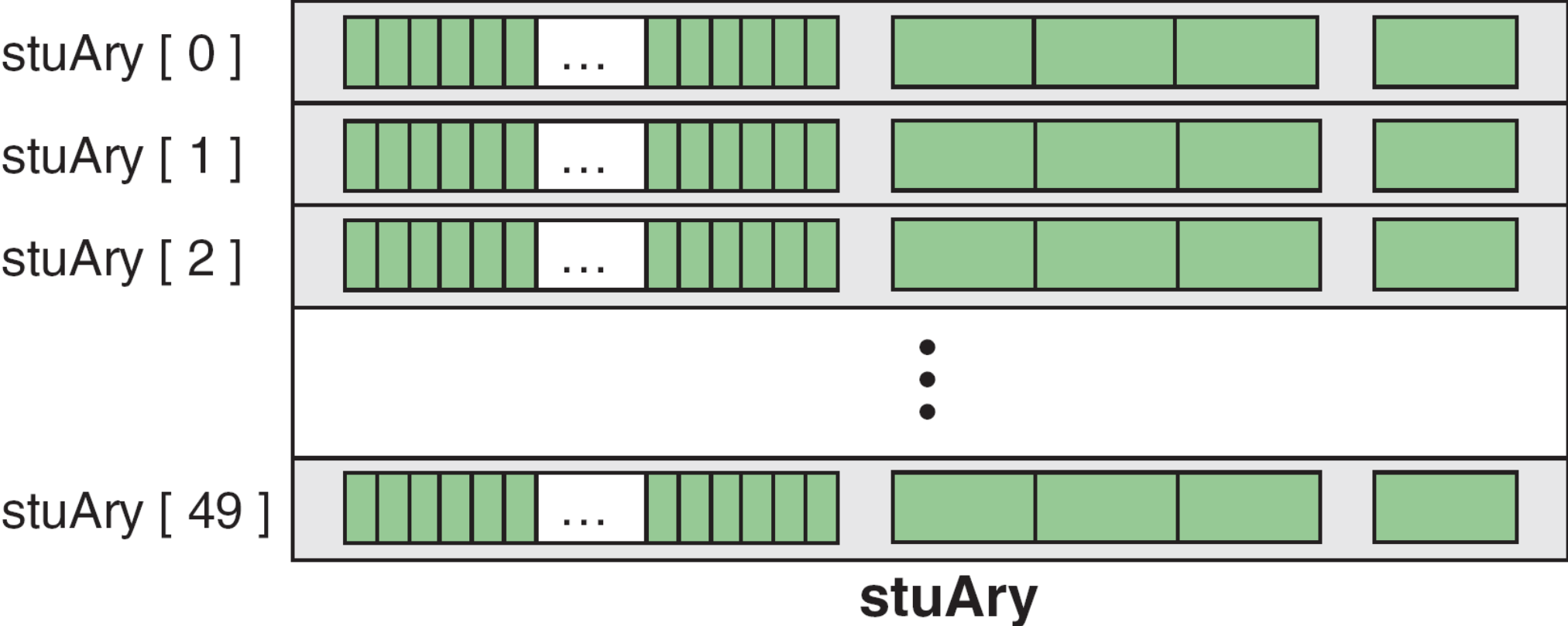


Arrays in Structures

```
// Global Declarations
typedef struct
{
    char name[26];
    int  midterm[3];
    int  final;
} STUDENT ;
// Local Declarations
STUDENT student;
```



Array of Structures



Example

- 이름, 나이, 성별(enum), 학년(enum)을 포함하는 구조체
- 크기 5짜리 구조체 배열 만들고 여기 있는 사람에 대한 정보 하나씩 넣기
- for 문으로 각 구조체에 들어간거 전부 출력

학년

FRESHMAN

SOPHOMORE

JUNIOR

SENIOR

성별

MALE

FEMALE

오늘 할 것

- Enumeration, Union

Enums

Enumerate Type

- 항목 혹은 선택지, 일종의 상수 항목을 만들 때 사용
- 각 항목들에 대해 identifier로서 하나의 정수(enumeration constant)가 할당됨
- 상황) 메뉴를 만들고 숫자 하나 입력받아서 switch 문으로 각기 다른 함수를 실행해야 한다.
- 그냥 숫자 0, 1, 2, 3으로 나누면 나중에 헷갈리고 불편함

Declaring an Enumerated Type

- `enum typeName {identifier list};`
- `enum color {RED, BLUE, GREEN, WHITE};`
- `enum color skyColor;`

Initializing Enumerated Constants

- `enum months {JAN, FEB, MAR, APR};`

identifier 자동 배정 0 1 2 3

- `enum months {JAN = 1, FEB, MAR, APR};`

- `enum color {RED, ROSE = 0, CRIMSON = 0, BLUE, AQUA = 1};`

Operations on Enumerated Types

- `enum color x, y;`
`x = BLUE; x = y;`
- `if (color1 == color2)`
`if (color1 == BLUE)`
- `Switch (color1)`
`{`
`case BLUE: ...`

Enumeration Type Conversion

- int형과 자동으로 호환됨
- `int x; enum color y;`
`x = BLUE; y = 2;`
- `enum color y;`
`y = (enum color) 2;`

Anonymous Enumeration: Constants

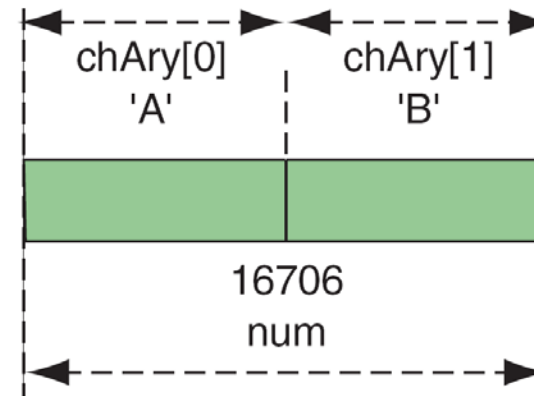
- `enum {space = ' ', comma = ',', color = ':'};`
- `enum {OFF, ON};`

Unions

Union Types

- 구조체와 비슷하다. 선언 및 사용도 구조체 처럼
- 단, Element 들이 메모리를 **공유**
- 전체 구조체 크기 = 구조체 내부의 가장 큰 변수의 크기

```
union shareData
{
    char    chAry[2];
    short   num;
};
```



Both num and chAry start at the same memory address. chAry[0] occupies the same memory as the most significant byte of num.

Example

```
typedef union {  
    short num;  
    char chary[2];  
} SHORTCHAR;
```

SHORTCHAR data;

data.num = 16706; // 1000001 01000010 = 65 66

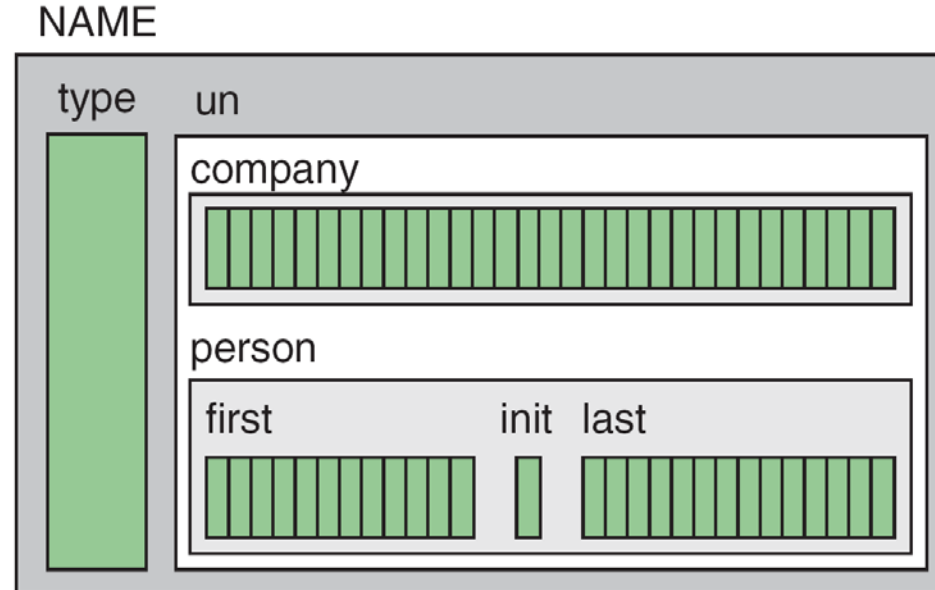
```
printf("Short: %hd\n", data.num);  
printf("Ch[0]: %c\n", data.chAry[0]);  
printf("Ch[1]: %c\n", data.chAry[1]);
```

Results:

```
Short: 16706  
Ch[0]: A  
Ch[1]: B
```

Union Application

```
typedef struct
{
    char  first[20];
    char  init;
    char  last[30];
} PERSON;
typedef struct
{
    char  type;
    union
    {
        char  company[40];
        PERSON person;
    } un;
} NAME;
```



메모리 공간 절약할 수 있고, 두 가지 구조체 역할을 할 수 있으므로 간단
단, 헷갈리지 않는다면

다음시간

- Various Data Structures implemented with Lists