전산 SMP 9주차

2015. 11. 24

김범수
bskim45@gmail.com

Special thanks to 박기석 (kisuk0521@gmail.com)

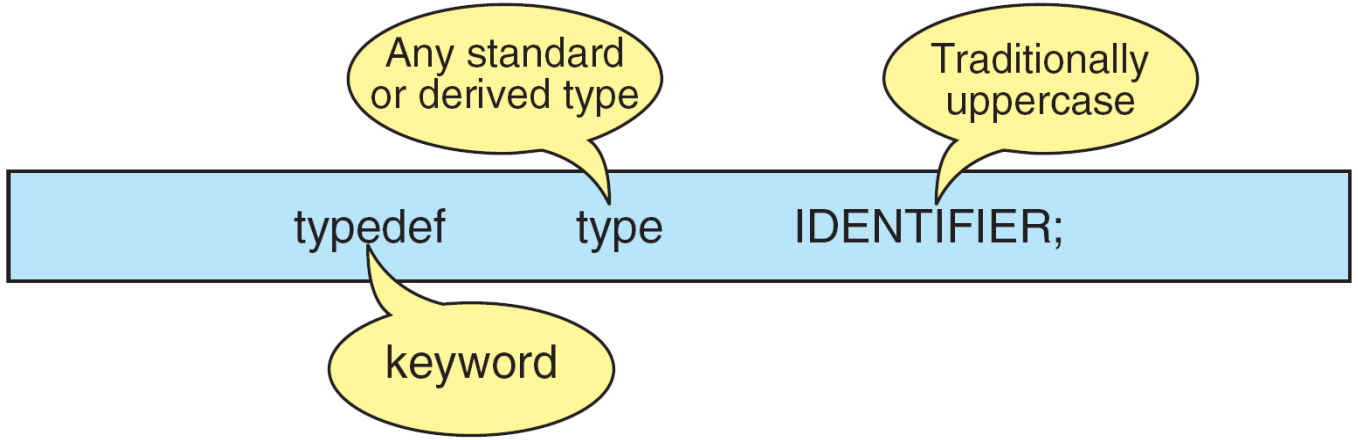
오늘 할 것

- Enumeration, Structure, Union

Structure

The Type Definition (typedef)

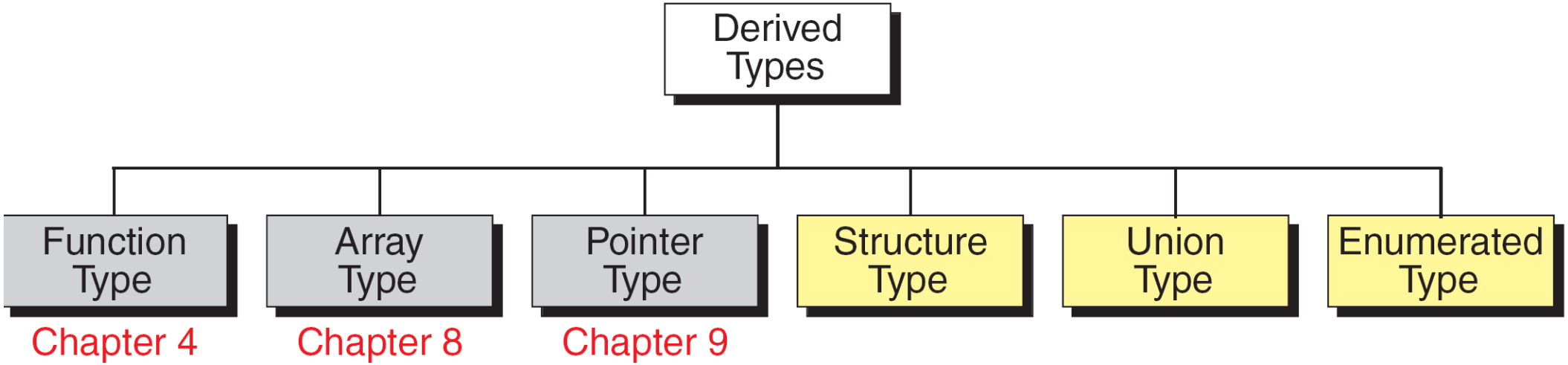
- A type definition, typedef, gives a name to a data type by creating a new type that can then be used anywhere a type is permitted.
- 프로그래머는 길게 쓰는 것을 싫어한다




```
typedef int INGETER;
```

Derived Type

- 프로그래머가 임의로 만든 타입



Enumerate Type

- 항목 혹은 **선택지**, 일종의 상수 항목을 만들 때 사용
- 각 항목들에 대해 identifier로써 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 COLORS {RED, ROSE=0, CRIMSON=0, BLUE, AQUA=1};
```

Operations on Enumerated Types

- `enum COLORS color1, color2;`

  `color1 = BLUE;`

  `color2 = color1;`

- `if (color1 == color2)`
  `if (color1 == BLUE)`

- `switch (color1) {`
  `case BLUE: …`

Enumeration Type Conversion

- int형과 자동으로 호환됨

- ```
  int x; enum COLORS y;
  x = BLUE; y = 2;
  ```

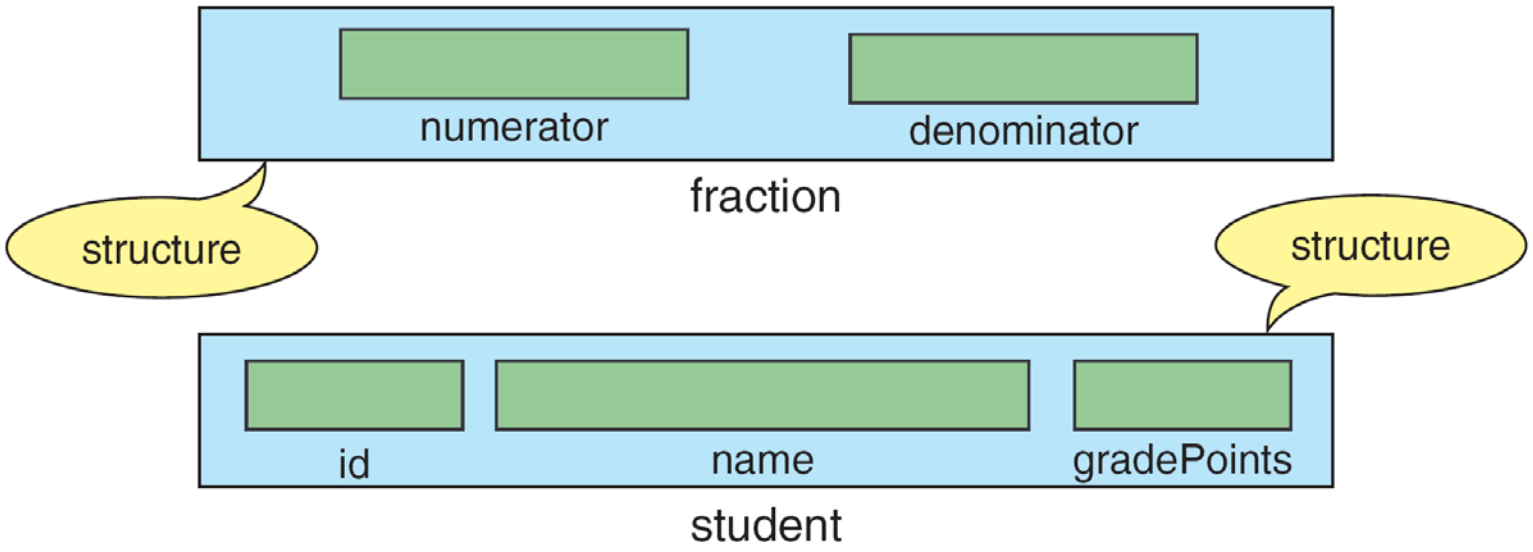
- ```
  enum COLORS y;
  y = (enum COLORS) 2;
  ```

Anonymous Enumeration: Constants

- `enum {space = ‘ ‘, comma = ‘,’, colon = ‘:’};`
- `enum {OFF, ON};`

Structure Type

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



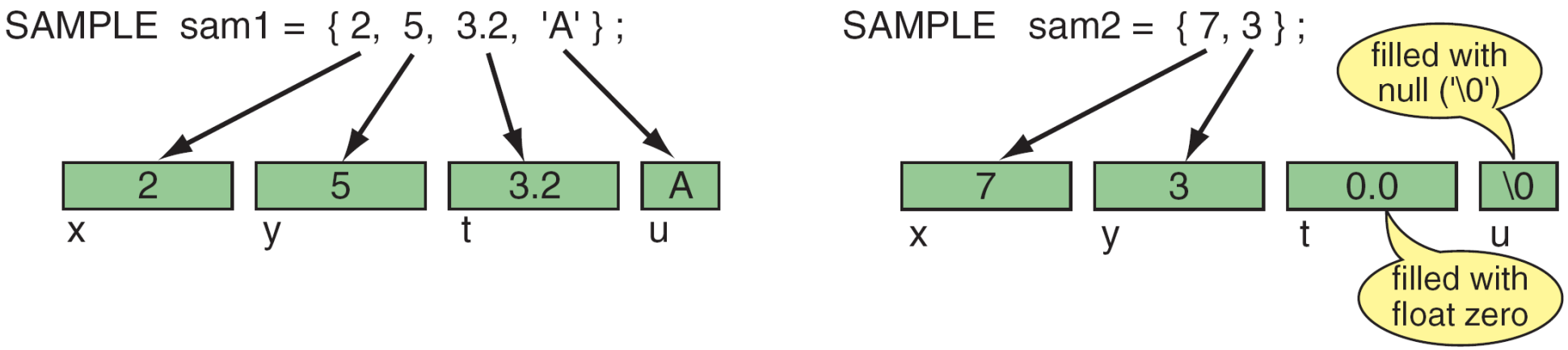
Structure Declaration

```
// 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;
```

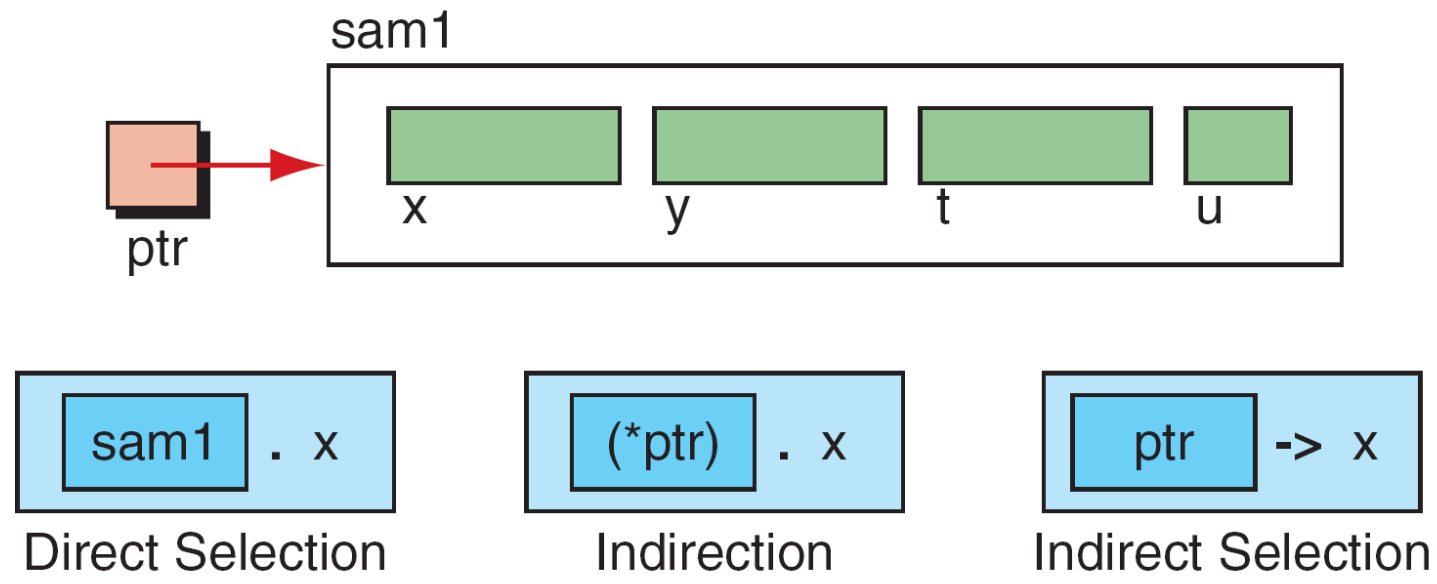
Initializing Structures

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



Accessing Structure Elements

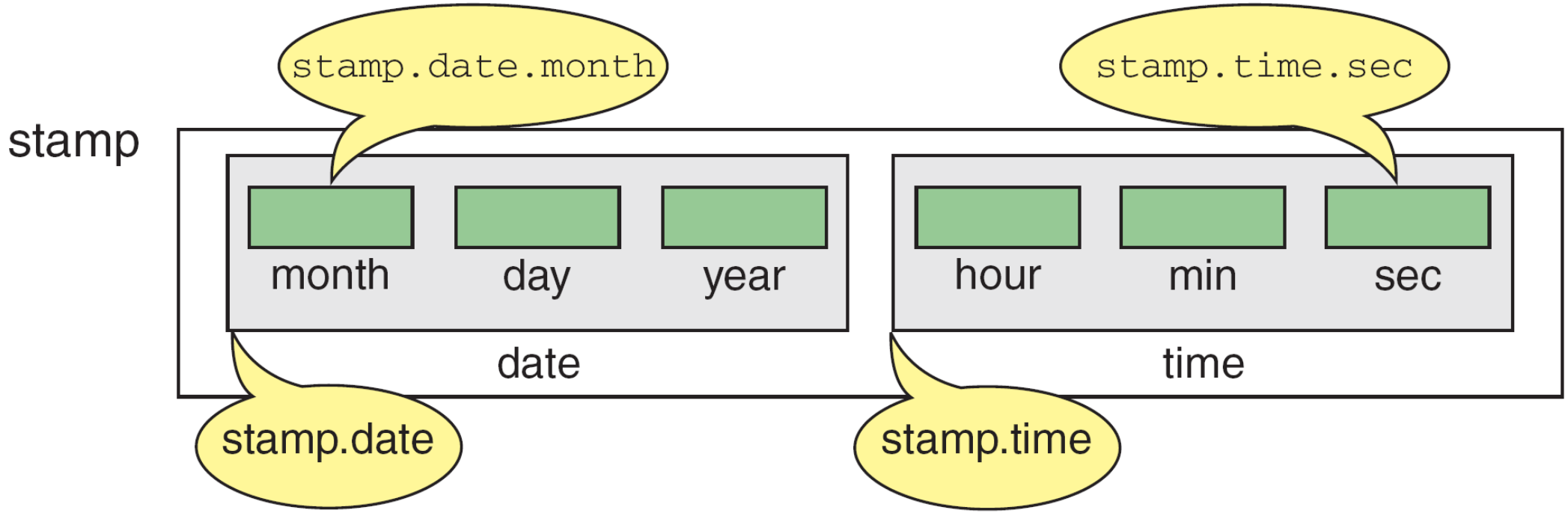
```
typedef struct
  {
   int    x;
   int    y;
   float  t;
   char   u;
  } SAMPLE;
  …
  SAMPLE  sam1;
  SAMPLE* ptr;
  …
  ptr = &sam1;
  …
```



Three Ways to Reference the Field **x**

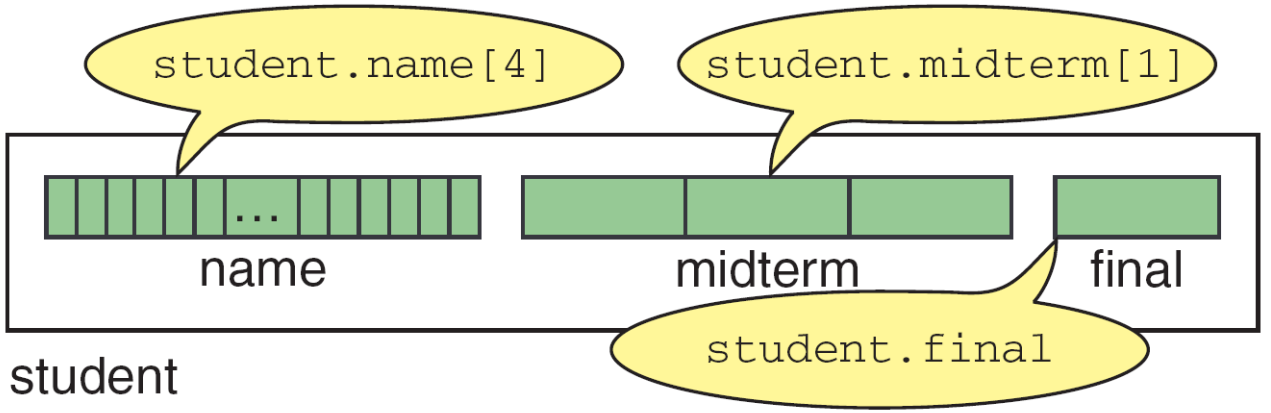
`(*pointerName).fieldname` ⟺ `pointerName->fieldName.`

Nested Structure

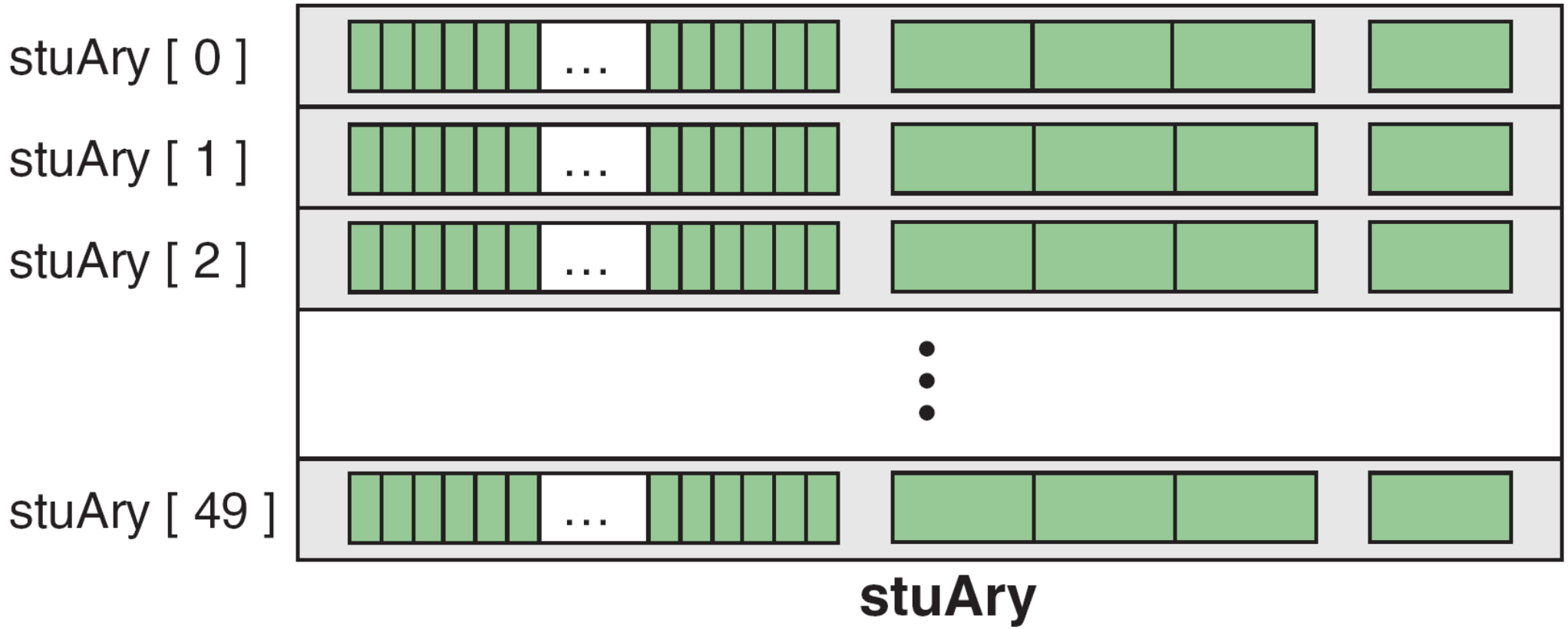


Arrays in Structures

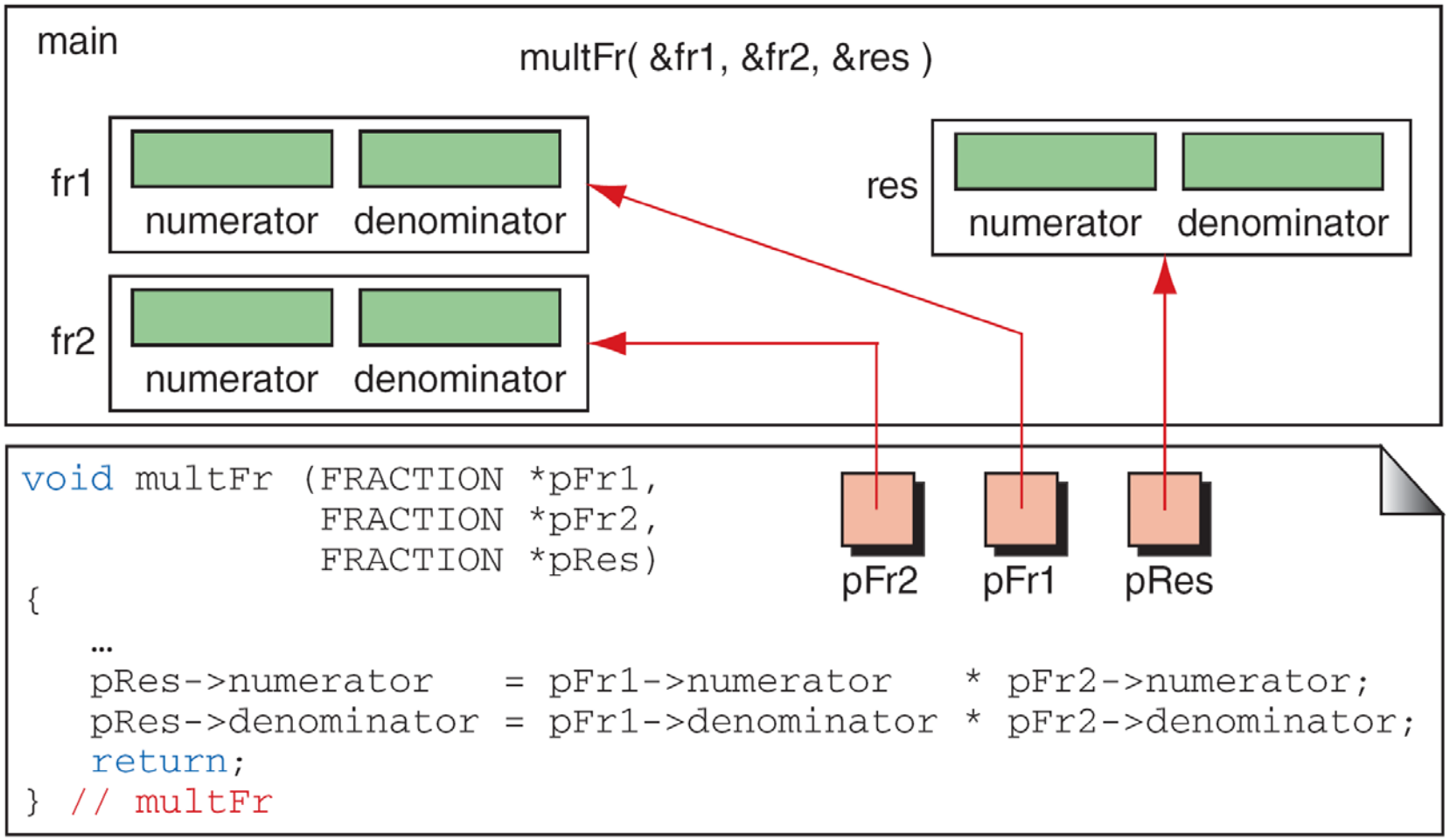
```
// Global Declarations
typedef struct
   {
    char name[26];
    int  midterm[3];
    int  final;
   } STUDENT ;
// Local Declarations
STUDENT student;
```



Array of Structures



Passing Structures Through Pointers



Structure Application

- Structure 배열 `struct AA ary[10];`
- Structure 포인터 배열 `struct AA *ary[10];`

- 한 Structure 안의 포인터 element가 다른 Structure를 가리킨다

→ Linked List의 핵심 개념!!

Example

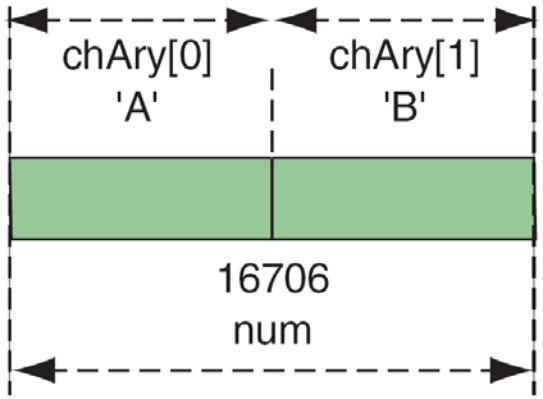
- 이름, 나이, 성별(enum), 학년(enum)을 포함하는 구조체
- 크기 5짜리 구조체 배열 만들고 여기 있는 사람에 대한 정보 하나씩 넣기
- for 문으로 각 구조체에 들어간거 전부 출력

학년	**성별**
FRESHMAN	MALE
SOPHOMORE	FEMALE
JUNIOR	
SENIOR	

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

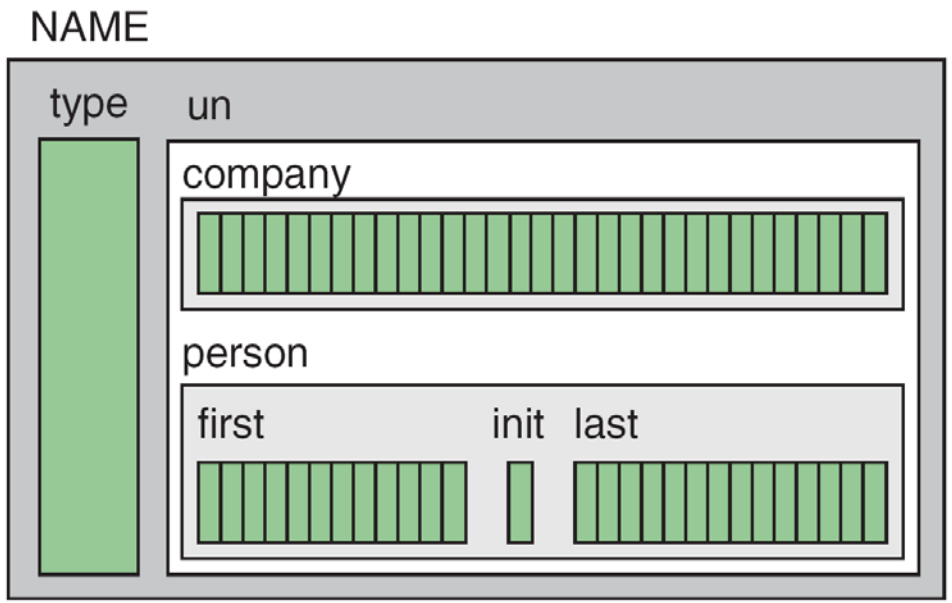
```
typedef union {
        short num;
        char  chary[2];
} SHORTCHAR;

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 공간을 절약할 수 있고, 두 가지 구조체 역할을 할 수 있으므로 간단

단, 헷갈리지 않는다면

Assn #4 – Prob 1: String 처리 함수 구현하기

```
<mystring.h>
int mystrlen(char *str);
char *mystrcpy(char *toStr, char *fromStr);
char *mystrcmp(char *str1, char *str2);
int mytolower(char *str);
```

- 저번 주에 기본적인 원리는 다 했죠? ^^

Assn #4 – Prob 2: 영한 사전 프로그램

- 〈string.h〉를 쓸 수 없다! Problem 1을 먼저 하고, 여기서 만든 mystring.h 를 include 하여 사용
- 5 개의 명령어(**single**, **multi**, **recent**, **quit**, **help**)는 별도의 함수로
- 단어 정보 구조체(WORDINFO), 그리고 단어의 **필요한 길이**만큼 메모리를 동적 할당을 받아 사용
  - WORDINFO 구조체를 동적 할당하고 → 단어와 단어의 뜻을 동적 할당
  - 쓰고 나서 반드시 free
- 예외 처리 – 파일 읽기에서 읽을 파일이 존재하지 않을 경우: fopen의 “r” 모드 리턴 값 NULL
- 파일 읽기 – fscanf가 NULL을 리턴할 때까지(EOF = -1)

Assn #4 - Prob 2: 영한 사전 프로그램

```
typedef struct{
    int num; // 몇 번째 단어
    char *word; // 영어 단어
    char *meaning; // 단어의 뜻(한글)
} WORDINFO;
WORDINFO recentTable[5];
```

1. single

word.txt을 위에서부터 쭉 돌면서 일치하는 단어가 나오면 멈추기. Super easy!

2. multi

파일 이름을 받아서 열고 위에서부터 쭉 돌면서 일치하는 단어가 있는지 확인!

앞에서 만든 single을 이용해서 짤 수 없을까?

3. recent

가장 최근의 단어가 앞에 나와야 함

단어를 검색할 때마다 리스트 맨 앞에 넣어주는 함수 만들기

Assn #4 – Prob 3: Hashed Dictionary

Hash

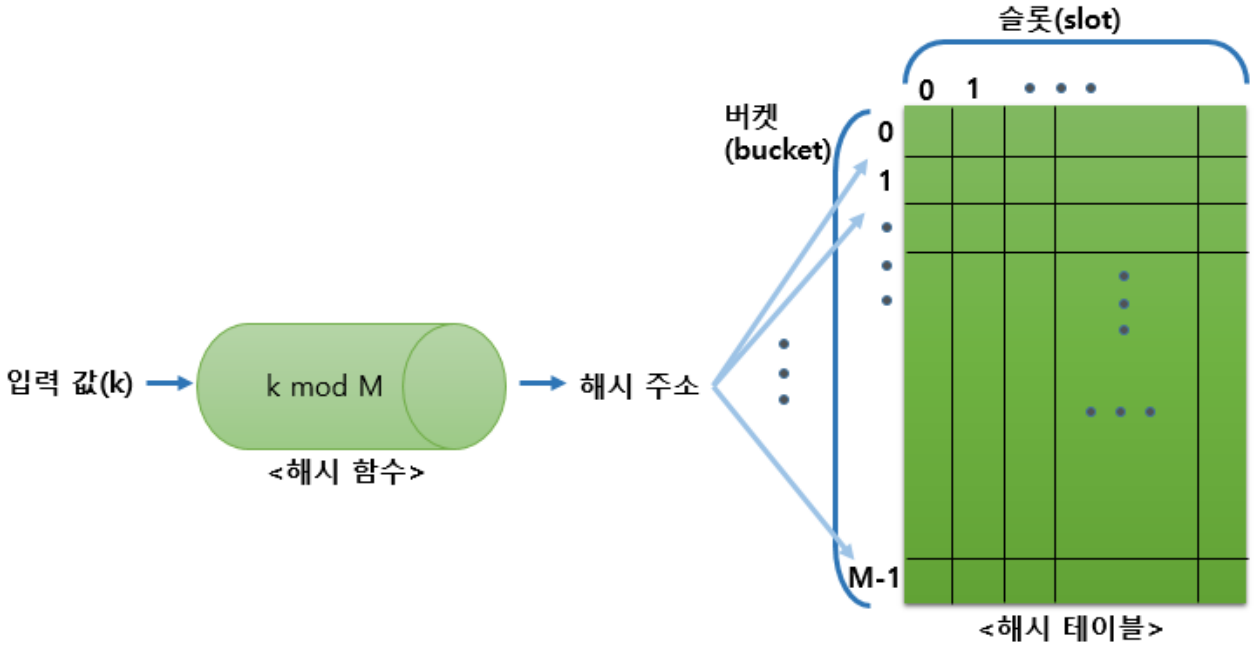
- 많은 양의 데이터를 효율적으로 처리하기 위한 자료구조
- 데이터가 너무 많으면 어떻게 찾을 거야?
- Hash function: h(k) = 데이터가 위치한 index – 수 많은 데이터를 작은 정수로 mapping

Hash **key**

Hash **function**

Hash **Address (Index)**

Hash **Table**

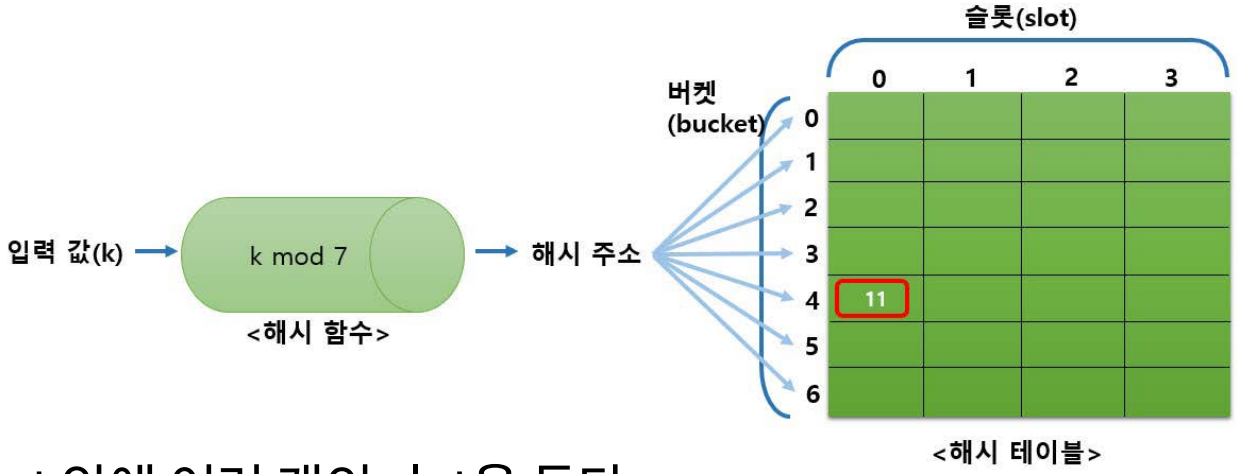


Assn #4 – Prob 3: Hashed Dictionary

- h(k) = k mod 7 → 버켓 7개

ex. k=11일 때 → h(11) = 11 mod 7 = **4**

→ **index가 4인 곳**에 저장

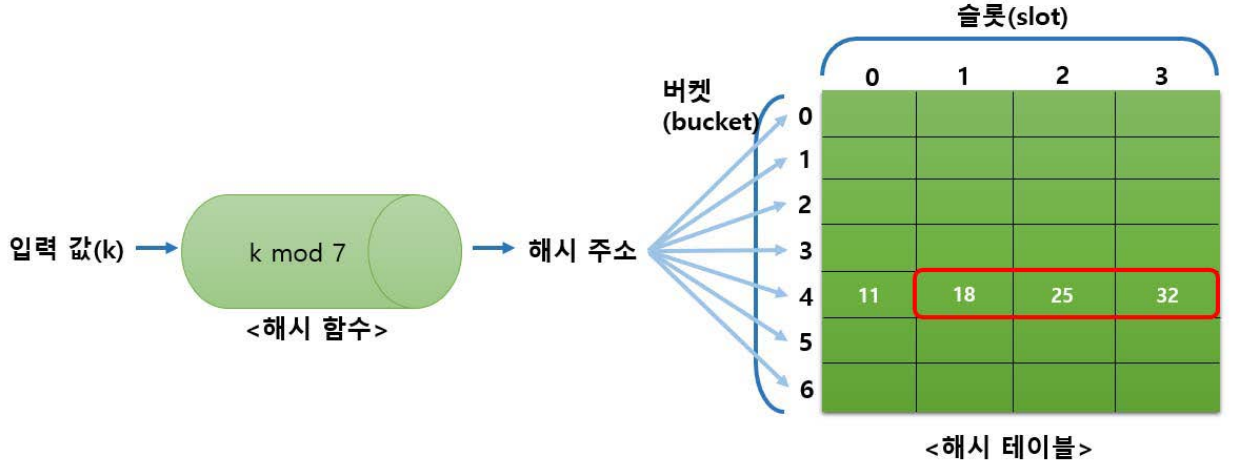


- **Collision**: 같은 hash 값을 가질 때 → bucket 안에 여러 개의 slot을 둔다

ex. h(18) = h(25) = h(32) = **4**

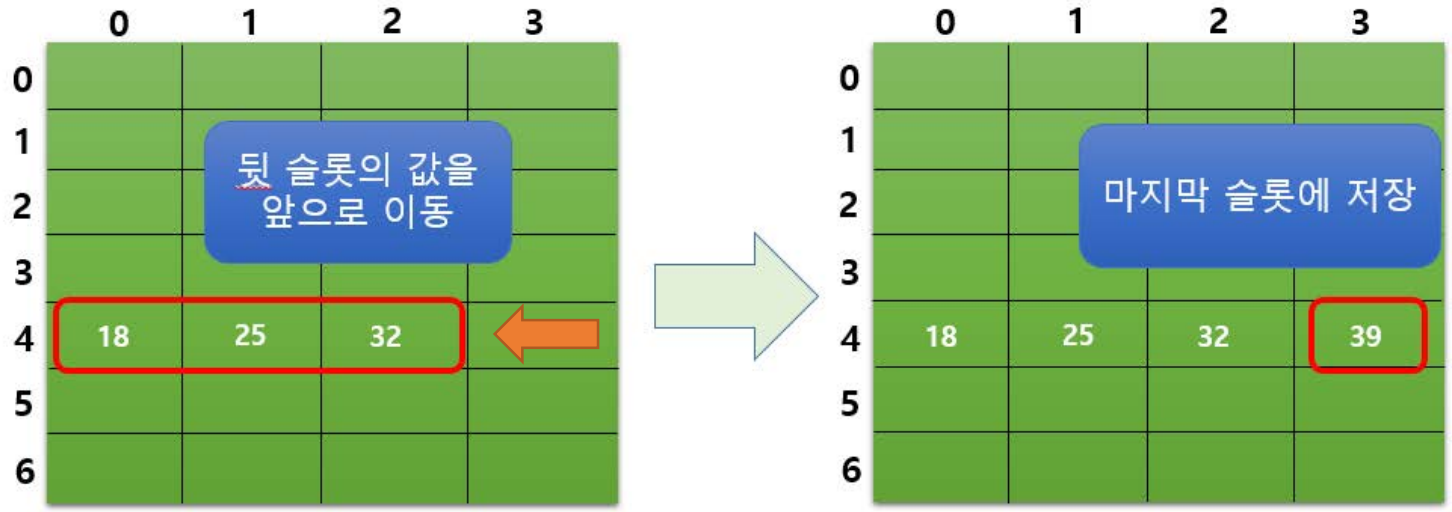
지금 상태에서 39를 저장하려고 한다면?

- **Overflow**: bucket이 꽉 찼을 때 → 어캄?



Assn #4 – Prob 3: Hashed Dictionary

- **Collision Handling**: collision이 발생했을 때 이를 해결하는 방법
- 오래 있었던 녀석 밀어내기(LRU): 오랫동안 안 쓰였다는 것은 앞으로도 안 쓰일 가능성이 높다
- bucket 4에 가장 오래 저장 되어 있었던 11을 삭제하고, 뒤 슬롯의 값을 앞으로 한 칸씩 이동 후, 제일 마지막 슬롯에 39를 저장한다.



Assn #4 – Prob 3: Hashed Dictionary

- 이제 Hash를 이번 어싸인에 적용시켜 보자!
- 자주 쓰는 단어를 Hash table에 저장해 놓고 사전을 찾기 전에 Hash table을 먼저 확인하기
- 있으면 좋고, 없으면 사전을 보면 됨!

1. 검색할 단어 입력.
2. **해시 테이블에 단어가 존재하는지 확인**.
3. 있다면, 해시 테이블에 저장된 단어 출력.
4. 없다면, 사전의 내용을 검색하여 출력하고, **해시 테이블에 검색된 단어 저장**.

Assn #4 – Prob 3: Hashed Dictionary

Hash 만들기

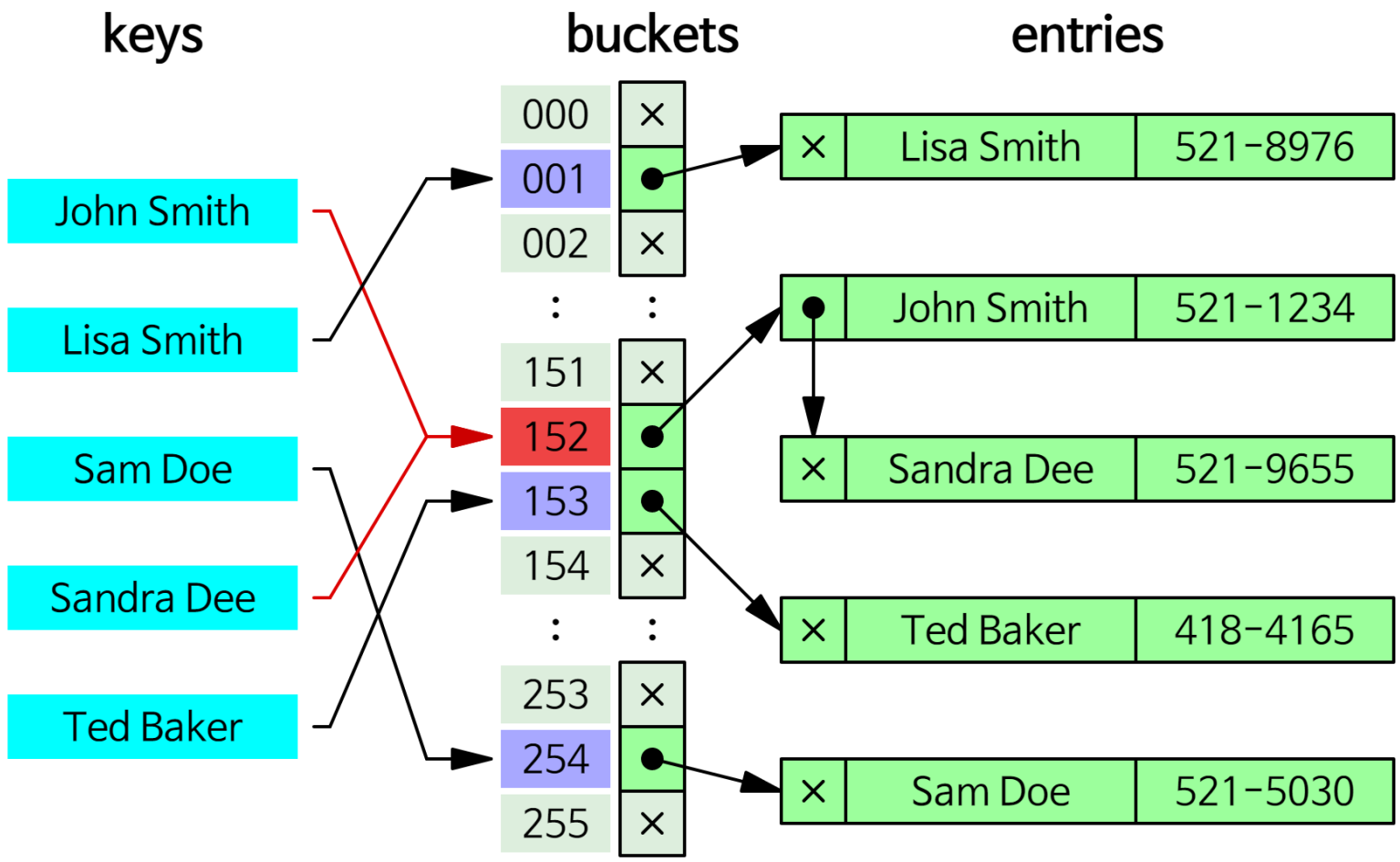
- bucket은 7개, slot는 4개. `WORDINFO`구조체 **동적 할당** 해서 저장
- h(k) = k mod 7
- `WORDINFO *hashTable[MOD_VAL][MAX_COL];` `// main 함수에 선언`

Key 정하기

- Hash를 사용하려면 키(k)를 알아야 하는데 단어는 숫자가 아니잖아?
- Key = 영어 단어의 아스키코드 값의 합

ex. “fat” → k = (102 + 97 + 116) = 315 → h(315) = 315 mod 7 = **0**

Assn #4 – Prob 3: Hashed Dictionary



Assn #4 – Prob 3: Hashed Dictionary

- 시나리오 #1: hash에 없음
- 시나리오 #2: hash에 있음
- 시나리오 #3: 오버플로우

Assn #4 – Prob 3: Hashed Dictionary

시나리오 #1: hash에 없음

```
>> single
검색할 단어: sharp
[검색 성공(파일)] 5.sharp : 날카로운
[저장 완료(해시 버켓 3)] 5.sharp : 날카로운
>>
```

```
>> print
버켓 0:
버켓 1:
버켓 2:
버켓 3: 5.sharp : 날카로운
버켓 4:
버켓 5:
버켓 6:
>>
```

시나리오 #2: hash에 있음

```
>> single
검색할 단어: sharp
[검색 성공(해시)] 5.sharp : 날카로운
>>
```

Assn #4 – Prob 3: Hashed Dictionary

시나리오 #3: overflow

```
>> print
버켓 0: 2.wet : 젖은  10.fat : 살찐  11.thirsty : 목마른  14.famous : 유명한
버켓 1:
버켓 2:
버켓 3: 5.sharp : 날카로운
버켓 4:
버켓 5:
버켓 6:
>>
```

```
>> single
검색할 단어: strange
[검색 성공(파일)] 21.strange : 이상한
[저장 실패(해시)] overflow
[삭제 완료(해시)] 2.wet : 젖은
[저장 완료(해시 버켓 0)] 21.strange : 이상한
>>
```

```
>> print
버켓 0: 10.fat : 살찐  11.thirsty : 목마른  14.famous : 유명한  21.strange : 이상한
버켓 1:
버켓 2:
버켓 3: 5.sharp : 날카로운
버켓 4:
버켓 5:
버켓 6:
>>
```

다음시간

- 어싸인 너무 쉽다
- Various Data Structures implemented with Lists