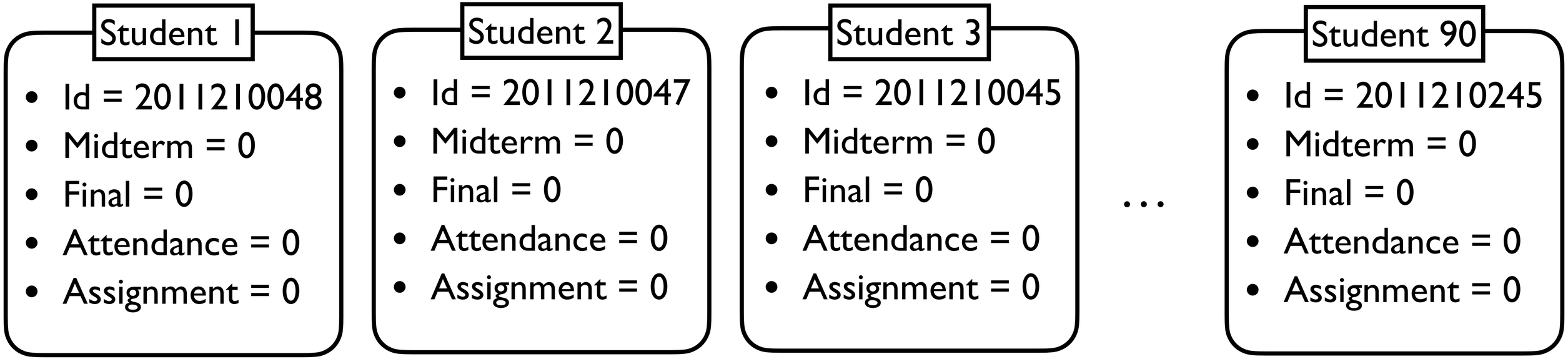
COSE213: Data Structure

Lecture 1 - 배열(Array)

Minseok Jeon

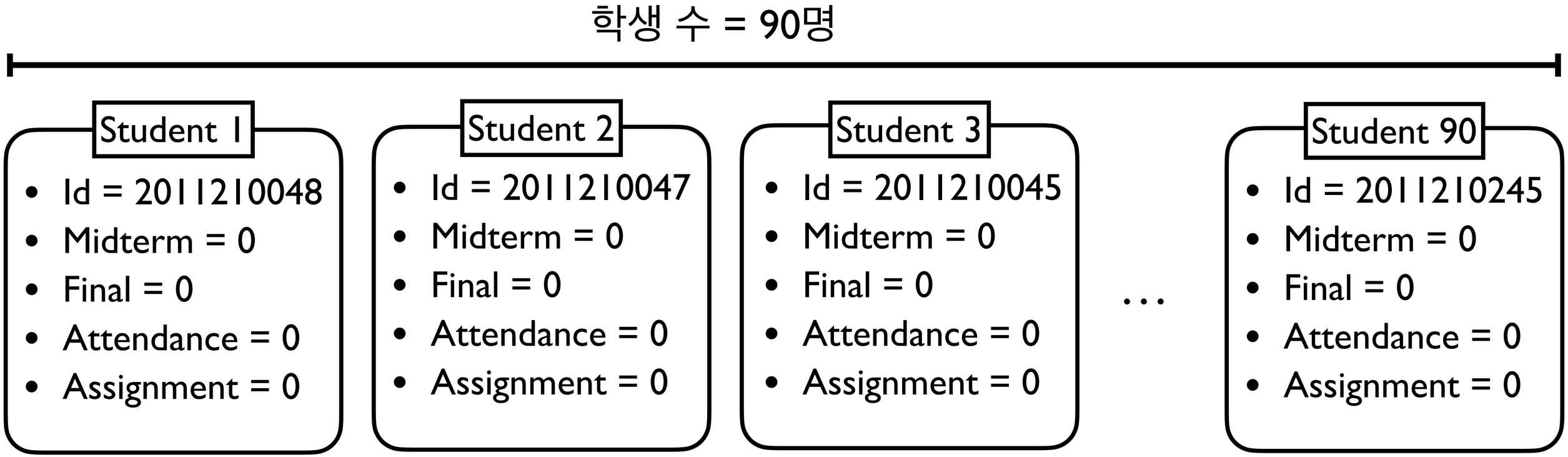
2024 Fall

문제: 학생 데이터 관리하기



문제: 학생 데이터 관리하기

- 특징1: 총 데이터 개수(학생 수)가 정해져 있음



문제: 학생 데이터 관리하기

- 특징1: 총 데이터 개수(학생 수)가 정해져 있음
- 특징2: 점수는 자주 업데이트 될 예정

Student 1

- Id = 2011210048
- Midterm = 58
- Final = 0
- Attendance = 0
- Assignment = 0

Student 2

- Id = 2011210047
- Midterm = 70
- Final = 0
- Attendance = 0
- Assignment = 0

Student 3

- Id = 2011210045
- Midterm = 47
- Final = 0
- Attendance = 0
- Assignment = 0

...

Student 90

- Id = 2011210245
- Midterm = 90
- Final = 0
- Attendance = 0
- Assignment = 0

문제: 학생 데이터 관리하기

- 특징1: 총 데이터 개수(학생 수)가 정해져 있음
- 특징2: 점수는 자주 업데이트 될 예정
- 특징3: 학생의 점수가 자주 탐색 될 예정

3번째 학생의 점수 탐색

Student 1

- Id = 2011210048
- Midterm = 58
- Final = 0
- Attendance = 0
- Assignment = 0

Student 2

- Id = 2011210047
- Midterm = 70
- Final = 0
- Attendance = 0
- Assignment = 0

Student 3

- Id = 2011210045
- Midterm = 47
- Final = 0
- Attendance = 0
- Assignment = 0

...

Student 90

- Id = 2011210245
- Midterm = 90
- Final = 0
- Attendance = 0
- Assignment = 0

문제: 학생 데이터 관리하기

- 특징1: 총 데이터 개수(학생 수)가 정해져 있음
- 특징2: 점수는 자주 업데이트 될 예정
- 특징3: 학생의 점수가 자주 탐색 될 예정
- 특징4: 순서가 생길 수 있음

Student 90
- Id = 20111210245
- Midterm = 90
- Final = 0
- Attendance = 0
- Assignment = 0

Student 2
- Id = 20111210047
- Midterm = 70
- Final = 0
- Attendance = 0
- Assignment = 0

Student 1
- Id = 20111210048
- Midterm = 58
- Final = 0
- Attendance = 0
- Assignment = 0

...

Student 33
- Id = 20111210045
- Midterm = 17
- Final = 0
- Attendance = 0
- Assignment = 0

자료구조 없이 데이터를 관리하는 경우

- 중간고사 점수의 평균 구하기

```
int main(){
  int mid_score1 = 85;
  int mid_score2 = 90;
  …
  int mid_score90 = 70;

  int mid_total = mid_score1 + mid_score2 + … + mid_score90;
  float mid_average = total/90;

  return 0;
}
```

자료구조 없이 데이터를 관리하는 경우

- 중간고사 점수의 중간값 구하기

```
int main(){
  int mid_score1 = 85;
  int mid_score2 = 90;
  …
  int mid_score90 = 70;

  return 0;
}
```

?

자료구조 없이 데이터를 관리하는 경우

- 중간고사 점수의 중간값 구하기 (점수가 5개인 경우)

```c
int main() {
    int a, b, c, d, e;

    printf("5개의 정수를 입력하세요:\n");
    scanf("%d %d %d %d %d", &a, &b, &c, &d, &e);

    if (a > b) swap(&a, &b);
    if (a > c) swap(&a, &c);
    if (a > d) swap(&a, &d);
    if (a > e) swap(&a, &e);

    if (b > c) swap(&b, &c);
    if (b > d) swap(&b, &d);
    if (b > e) swap(&b, &e);

    if (c > d) swap(&c, &d);
    if (c > e) swap(&c, &e);

    if (d > e) swap(&d, &e);

   printf("중간값: %d\n", c);

    return 0;
}
```

```c
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

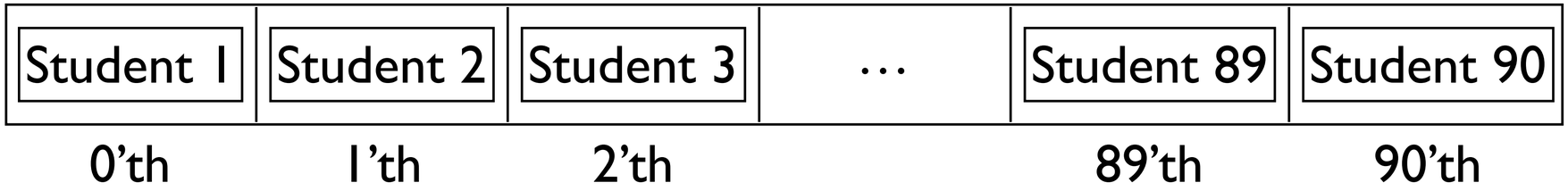
자료구조 없이 데이터를 관리하는 경우

- 중간고사 점수의 중간값 구하기 (변수가 9개인 경우)

```
int find_median(int a, int b, int c, int d, int e, int f, int g, int h, int i) {
    // 9개의 변수에 대해 정렬을 수행
    if (a > b) swap(&a, &b);
    if (a > c) swap(&a, &c);
    if (a > d) swap(&a, &d);
    if (a > e) swap(&a, &e);
    if (a > f) swap(&a, &f);
    if (a > g) swap(&a, &g);
    if (a > h) swap(&a, &h);
    if (a > i) swap(&a, &i);

    if (b > c) swap(&b, &c);
    if (b > d) swap(&b, &d);
    if (b > e) swap(&b, &e);
    if (b > f) swap(&b, &f);
    if (b > g) swap(&b, &g);
    if (b > h) swap(&b, &h);
    if (b > i) swap(&b, &i);

    if (c > d) swap(&c, &d);
    if (c > e) swap(&c, &e);
    if (c > f) swap(&c, &f);
    if (c > g) swap(&c, &g);
    if (c > h) swap(&c, &h);
    if (c > i) swap(&c, &i);
        if (d > e) swap(&d, &e);
        if (d > f) swap(&d, &f);
        if (d > g) swap(&d, &g);
        if (d > h) swap(&d, &h);
        if (d > i) swap(&d, &i);

        if (e > f) swap(&e, &f);
        if (e > g) swap(&e, &g);
        if (e > h) swap(&e, &h);
        if (e > i) swap(&e, &i);

        if (f > g) swap(&f, &g);
        if (f > h) swap(&f, &h);
        if (f > i) swap(&f, &i);

        if (g > h) swap(&g, &h);
        if (g > i) swap(&g, &i);

        if (h > i) swap(&h, &i);

        // 중간값을 반환 (정렬 후 5번째 값)
        return e;
    }
```

적합한 자료구조: 배열(Array)

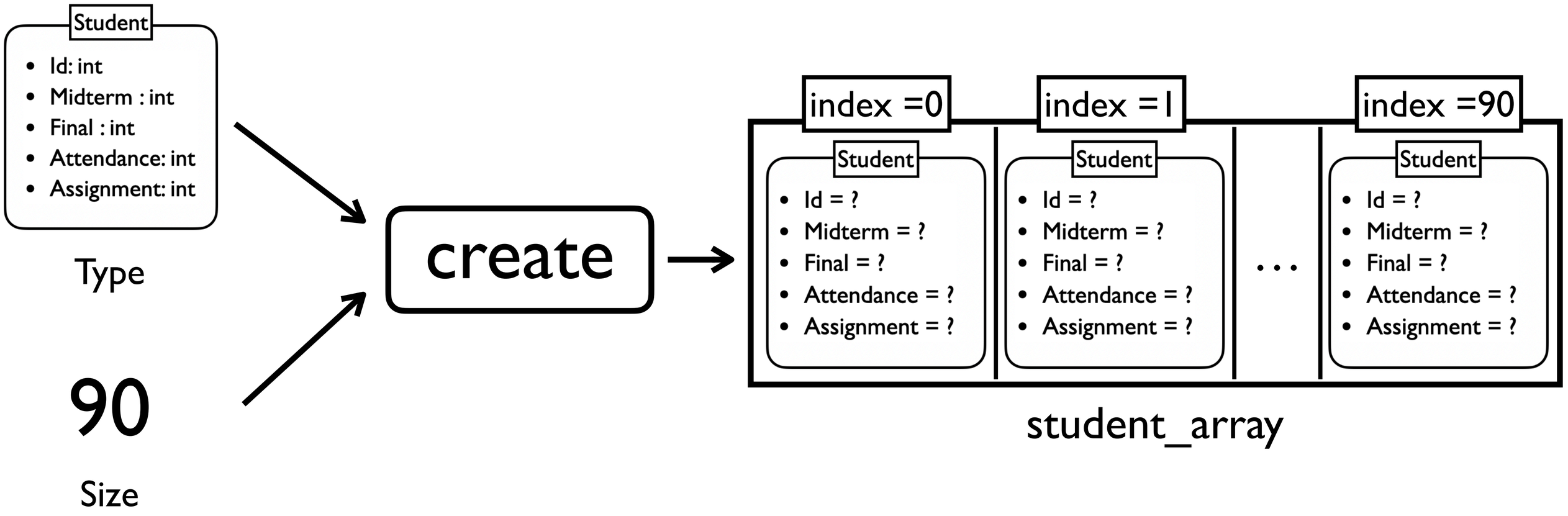
- 배열(Array)은 동일한 데이터 타입을 가진 값들을 연속된 공간에 저장하는 자료구조.



- 배열 자료구조는 다음의 기능들을 제공함
  - `create(type,size)` : 주어진 타입(`type`)과 길이(`size`)를 가지는 배열을 생성
  - `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환
  - `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장

Create

- `create(type,size)` : 주어진 타입(type)과 길이(size)를 가지는 배열을 생성



Create

- create(type,size) : 주어진 타입(type)과 길이(size)를 가지는 배열을 생성

```
typedef struct {
  int id;
  int midterm;
  int final;
  int attendance;
  int assignment;
} Student;
```

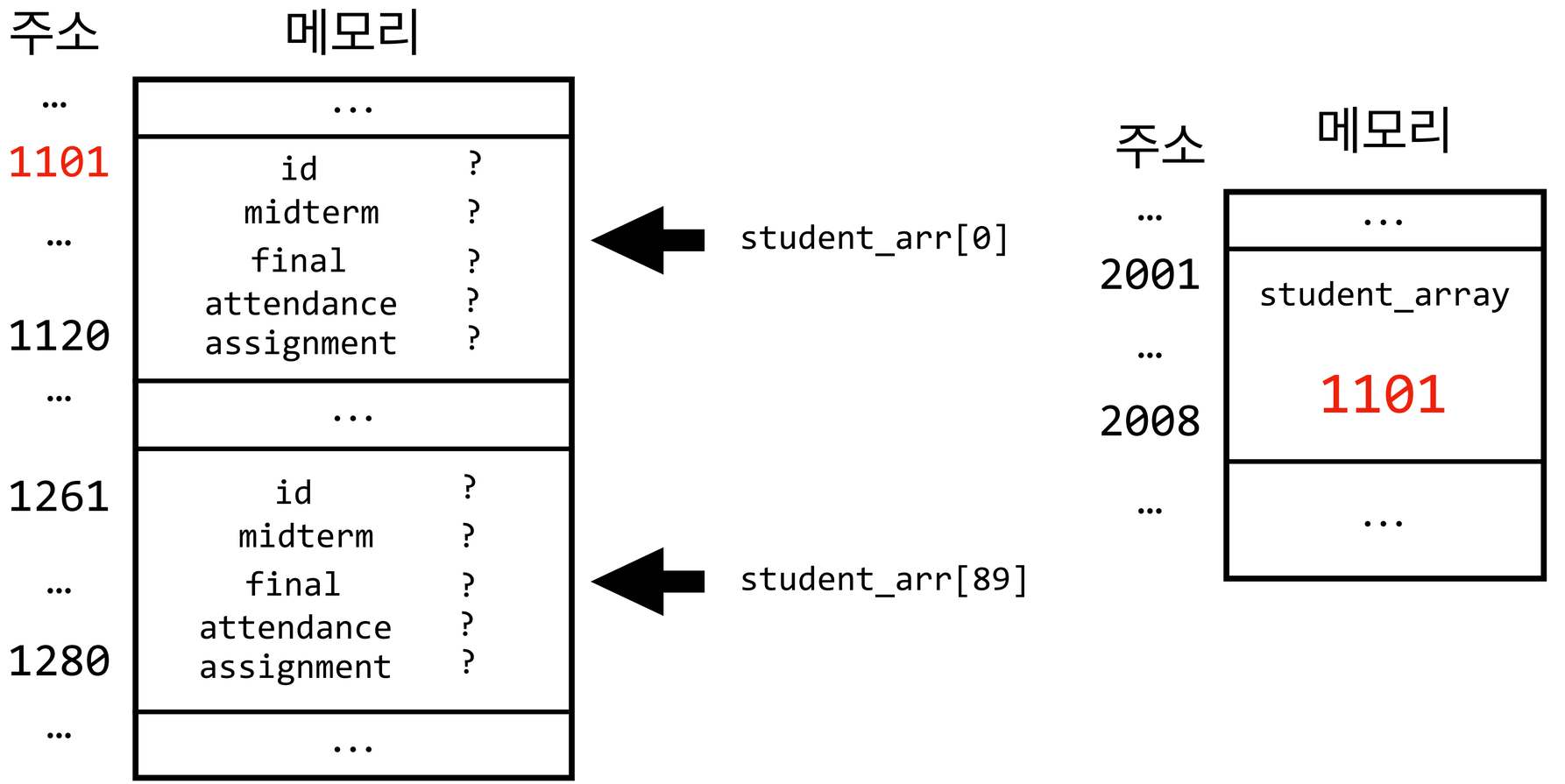
```
Student student_array[90];
```

- Student 타입의 길이 90인 배열

```
typedef struct {
  int id;
  int midterm;
  int final;
  int attendance;
  int assignment;
} Student;
```

Create

```
Student student_array[90];
```



Create

- create(type,size) : 주어진 타입(type)과 길이(size)를 가지는 배열을 생성

```
Student *student_array = (Student *) malloc(sizeof(Student) * 90);
```

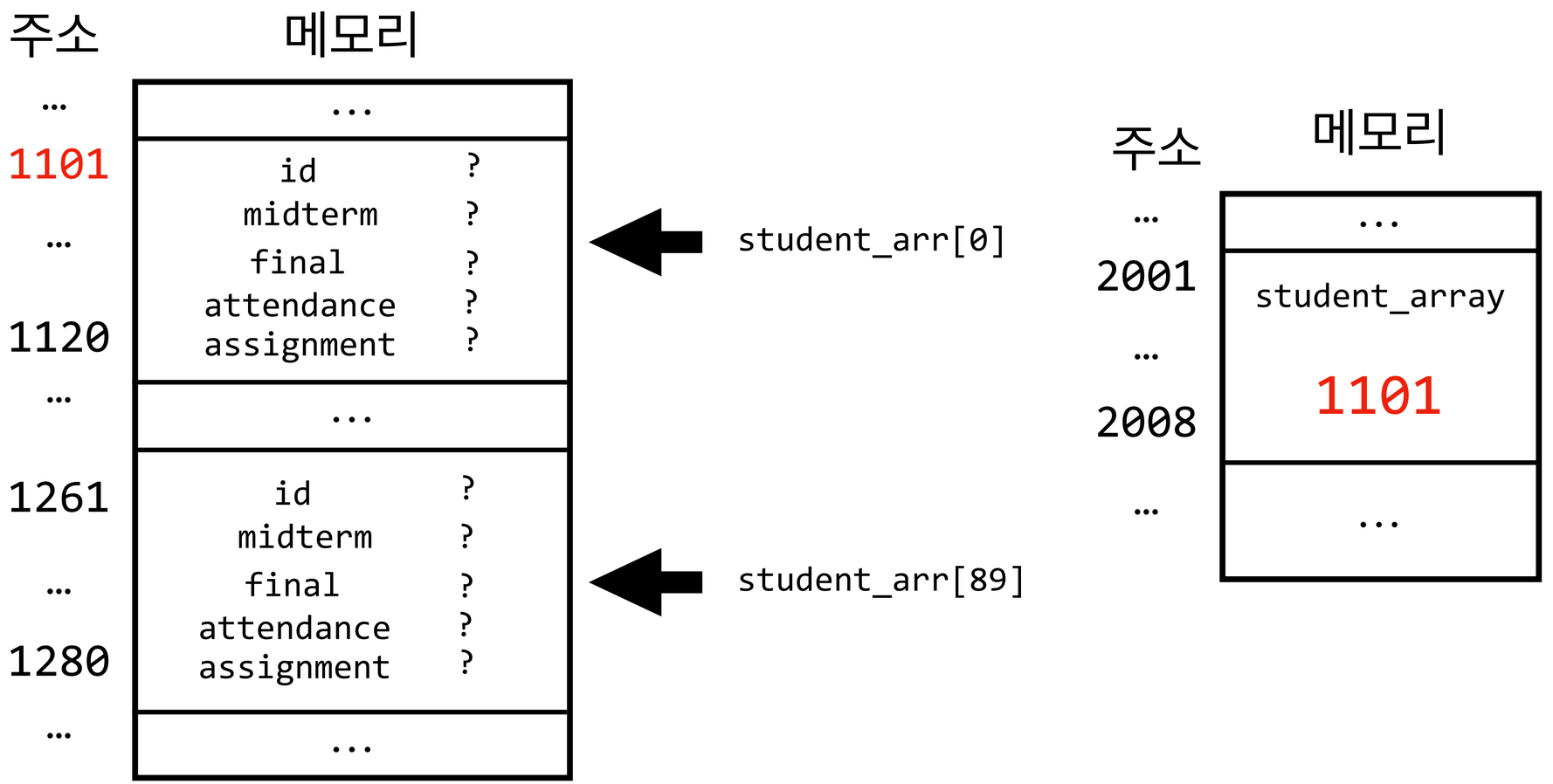
```
typedef struct {
  int id;
  int midterm;
  int final;
  int attendance;
  int assignment;
} Student;
```

- Student 타입의 길이 90인 배열

Create

```
Student *student_array = (Student *) malloc(sizeof(Student) * 90);
```

```
typedef struct {
  int id;
  int midterm;
  int final;
  int attendance;
  int assignment;
} Student;
```



Create

- create(type,size) : 주어진 타입(type)과 길이(size)를 가지는 배열을 생성

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

typedef struct {
  int id;
  int midterm;
  int final;
  int attendance;
  int assignment;
} Student;

int main(){
    Student student_array[90];
    printf("Size of int type : %lu Bytes\n", sizeof(int));
    printf("Size of Student type : %lu Bytes\n", sizeof(Student));
    printf("Size of student_array : %lu Bytes\n", sizeof(student_array));
    printf("Value of student_array : %p\n", student_array);
    printf("Address of student_array[0] : %p\n", &student_array[0]);
    printf("Address of student_array[0].id : %p\n", &student_array[0].id);

    Student *student_ptr = (Student *) malloc(sizeof(Student) * 90);
    printf("Initial midterm score of the first student : %d\n",student_ptr[0].midterm);
    free(student_ptr);
    // free(student_array); // error

    return 0;
}
```

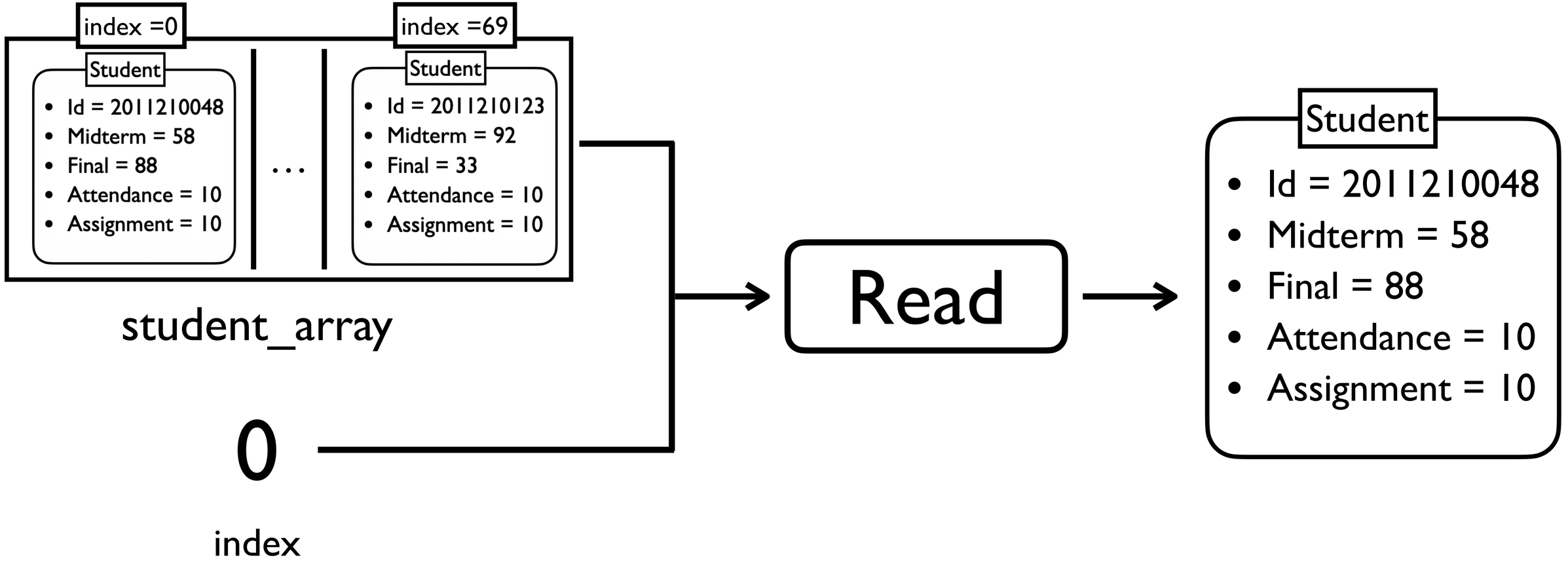
Create

- create(type,size) : 주어진 타입(type)과 길이(size)를 가지는 배열을 생성

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

typedef struct {
  int id;
  int midterm;
  int final;
  int attendance;
  int assignment;
} Student;

Student* f(){
    Student *student_array = (Student *) malloc(sizeof(Student) * 2);
    // Student student_array[2];
    return student_array;
}

int main(){
    Student *ptr = f();
    // free(ptr); // error
    printf("%d\n", ptr[0].id);
    return 0;
}
```

Read

- `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환



Read

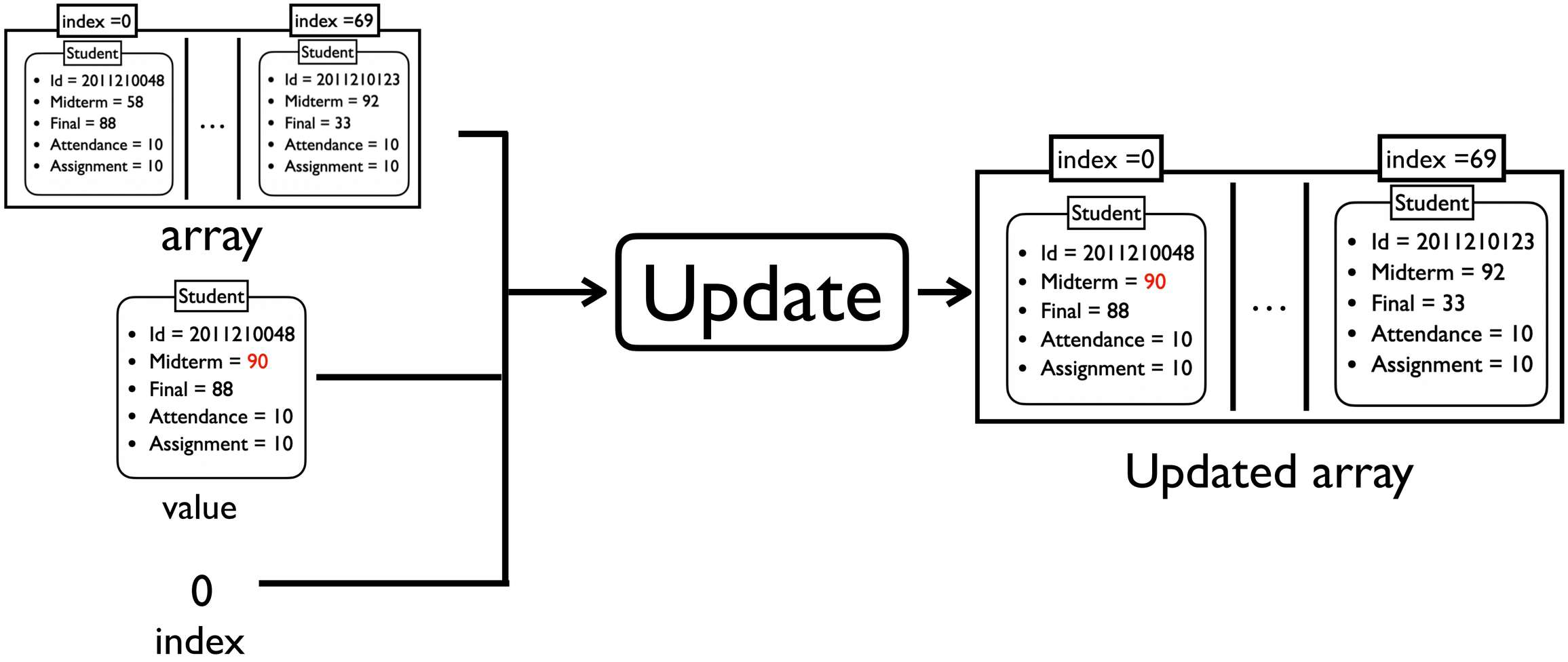
- `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환

```
student = student_array[0];
```

- student_array 배열에 0번째 데이터 접근 후 student에 저장함

Update

- `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장



Update

- `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장

```
student_array[0] = new_data;
```

- 배열의 첫번째 원소의 값을 new_data로 업데이트 함

Example

- 중간고사 점수의 중간값 구하기 (배열을 사용하는 경우)

```
int main() {
    int arr[90] = {85,…,70};
    int n = 90;
    sort(arr, n);
    int median = arr[n / 2];
    printf("중간값: %d\n", median);
    return 0;
}
```

```
void sort(int arr[], int n) {
    int i, j;
    for (i = 0; i < n-1; i++) {
        for (j = 0; j < n-i-1; j++) {
            if (arr[j] > arr[j+1]) {
                swap(&arr[j], &arr[j+1]);
            }
        }
    }
}
```

```
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
  }
```

배열(Array)의 사용 설명서

- 배열(Array)은 동일한 데이터 타입을 가진 값들을 연속된 공간에 저장하는 자료구조.
- 배열 자료구조는 다음의 기능들을 제공함:
  - `create(type,size)` : 주어진 타입(`type`)과 길이(`size`)를 가지는 배열을 생성
  - `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환
  - `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장
  - `size(arr)` : 배열의 크기를 반환함
  - `search(arr, value)` : 배열의 데이터 중 `value`의 인덱스(`index`)값을 반환함

  …

배열(Array)의 사용 설명서

- 배열(Array)은 동일한 데이터 타입을 가진 값들을 연속된 공간에 저장하는 자료구조.
- 배열 자료구조는 다음의 기능들을 제공함:
  - `create(type,size)` : 주어진 타입(`type`)과 길이(`size`)를 가지는 배열을 생성
  - `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환
  - `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장
  - `size(arr)` : 배열의 크기를 반환함
  - `search(arr, value)` : 배열의 데이터 중 `value`의 인덱스(`index`)값을 반환함

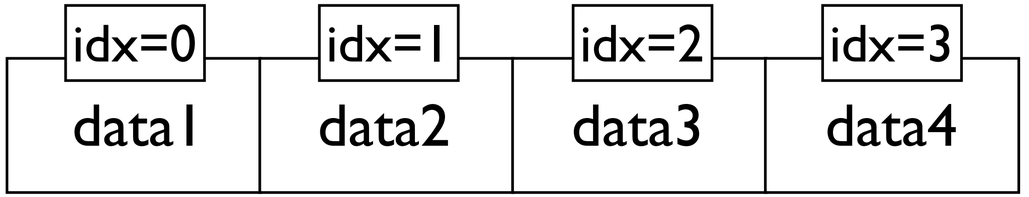
  …

배열의 **추상 자료형**(Abstract Data Type)

- ADT of array in Java: https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html
- ADT of array in Ocaml: https://ocaml.org/manual/5.2/api/Array.html

믹서기의 사용설명서

- 믹서기는 재료를 갈아주는 기계입니다.
- 믹서기는 아래 세가지 버튼으로 사용하면 됩니다.
    - mix 버튼: 내용물을 갊
    - stop 버튼: 멈춤



배열의 사용 설명서 (배열의 추상 자료형)

- 배열은 데이터를 연속된 공간에 저장하는 자료구조.
- 배열은 아래 4가지 기능을 제공함
    - `create` : 배열을 생성함
    - `read` : 주어진 인덱스에 해당하는 자료를 반환
    - `update` : 주어진 인덱스 위치에 새로운 데이터를 저장
    - `size` : 배열의 크기를 반환함

추상 자료형 (Abstract Data Type)

- 자료구조를 표현하는 대표적인 방법
  - 사용자의 관점에서 자료구조가 데이터를 처리하는 방법과 그에 대한 연산으로 자료구조를 표현
    - 사용자의 입장에서 불필요한 정보를 숨김
- 배열: 아래와 같은 연산을 제공하는 자료구조!
  - `create(type,size)` : 주어진 타입(`type`)과 길이(`size`)를 가지는 배열을 생성
  - `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환
  - `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장

- ADT of array in Java: https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html
- ADT of array in Ocaml: https://ocaml.org/manual/5.2/api/Array.html

Example: List (리스트)

- 리스트(List)는 크기가 고정되지 않고 필요에 따라 자동으로 크기가 늘어나는 자료구조
  - 임의의 위치에 읽기 및 수정 가능

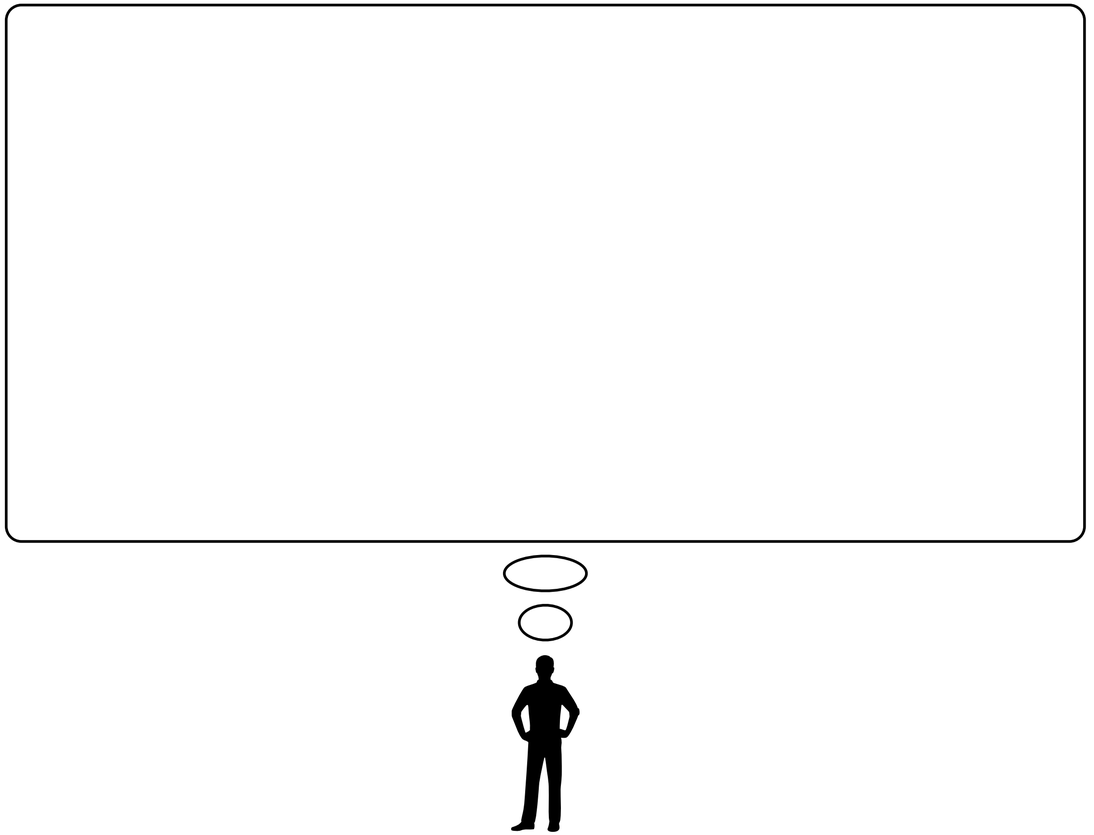
- 리스트의 추상 자료형 (Abstract Data Type):
  - `List* create()` : 빈 리스트를 생성 후 반환
  - `int read(List *arr, int index)` : 리스트에서 주어진 인덱스에 해당하는 정수값을 반환
  - `void update(List *arr, int index, int element)` : 리스트에서 주어진 인덱스 위치에 새로운 정수 데이터를 저장
  - `void append(List *arr, int element)`: 리스트 끝에 새로운 정수를 추가함. 가득찰 경우 리스트의 용량이 늘어남.
  - `void destroy(List *arr)` : 리스트가 차지하고있는 메모리를 해제함

Example

- ADT: 사용자의 관점에서 자료구조가 데이터를 처리하는 방법과 그에 대한 연산으로 자료구조를 표현

```c
#include “List.c”
int main() {
    List *lst = create();
    for (int i = 0; i<70;++i){
      append(lst, i);
    }
    update(lst, 0, 100);
    printf(“elements:\n");
    for (int i = 0; i < 70; i++) {
        printf("%d ", read(lst, i));
    }
    destroy(lst);
    return 0;
}
```

- 사용자의 관점



마무리 (Wrap-up)

- 문제:
  - 우리는 종종 고정된 길이와 타입을 가지는 데이터를 관리(접근 및 업데이트)해야함. (예시: COSE213 수강생 데이터 관리)
- 해결책:
  - 배열(Array): 동일한 데이터 타입을 가진 값들을 연속된 공간에 저장하는 자료구조.
  - 배열 자료구조는 다음의 기능들을 제공함 (배열의 추상 자료형):
    - `create(type,size)` : 주어진 타입(`type`)과 길이(`size`)를 가지는 배열을 생성
    - `read(arr,index)` : 배열(`arr`)에서 주어진 인덱스(`index`)에 해당하는 자료를 반환
    - `update(arr,index,value)` : 배열(`arr`)에서 주어진 인덱스(`index`) 위치에 새로운 데이터(`value`)를 저장
- 사용 예시:

create

```
Student arr[70];
```

read

```
Student student = arr[i];
```

update

```
arr[i] = value;
```

- 추상 자료형(Abstract Data Type):
  - 사용자의 관점에서 자료구조가 데이터를 처리하는 방법과 그에 대한 연산으로 자료구조를 표현

COSE213: Data Structure

Lecture 1-2 - C 언어 리뷰

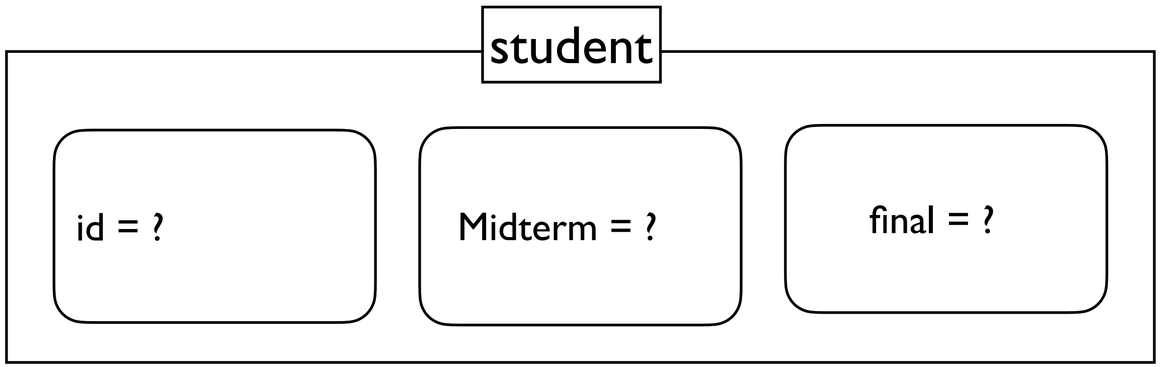
Minseok Jeon

2024 Fall

C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- “.”을 사용하여 데이터에 접근 및 업데이트 할 수 있음

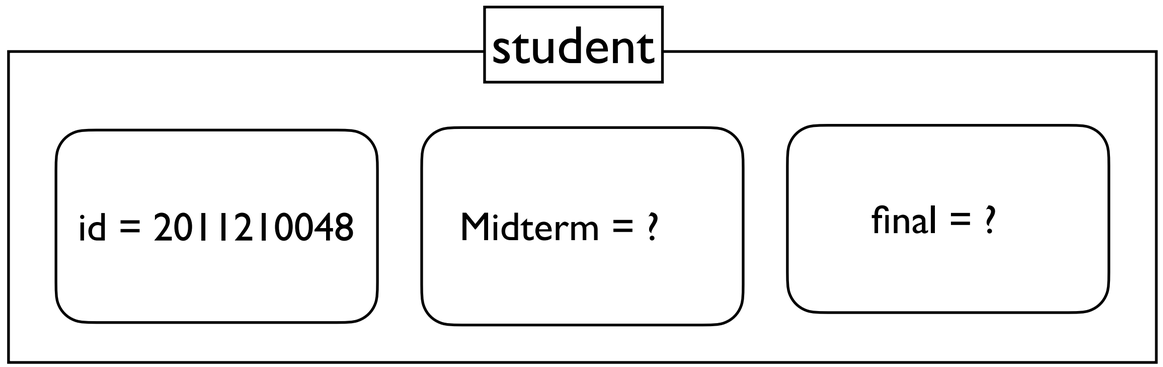
```c
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  student.id = 2011210048;
  student.midterm = 60;
  student.final = 70;
}
```



C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- “.”을 사용하여 데이터에 접근 및 업데이트 할 수 있음

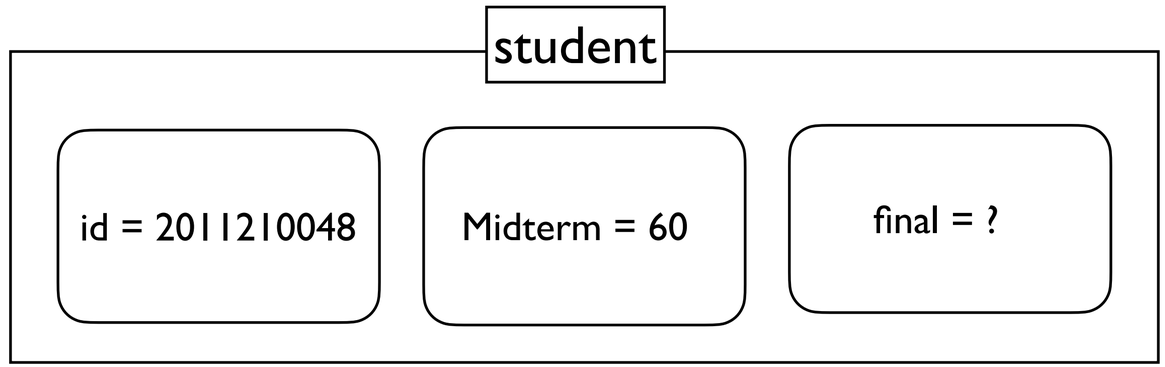
```c
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  student.id = 2011210048;
  student.midterm = 60;
  student.final = 70;
}
```



C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- “.”을 사용하여 데이터에 접근 및 업데이트 할 수 있음

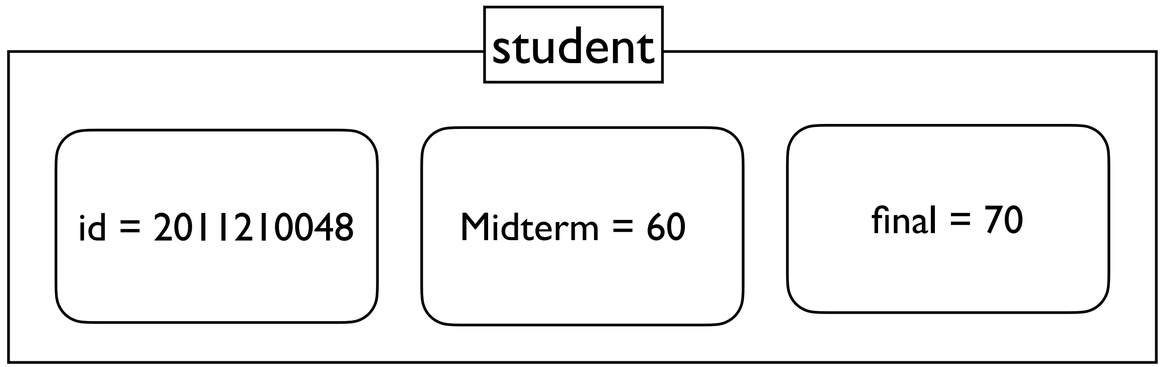
```c
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  student.id = 2011210048;
  student.midterm = 60;
  student.final = 70;
}
```



C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- “.”을 사용하여 데이터에 접근 및 업데이트 할 수 있음

```c
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  student.id = 2011210048;
  student.midterm = 60;
  student.final = 70;
}
```

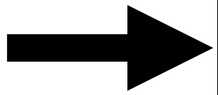


C 언어 리뷰 : 포인터 변수

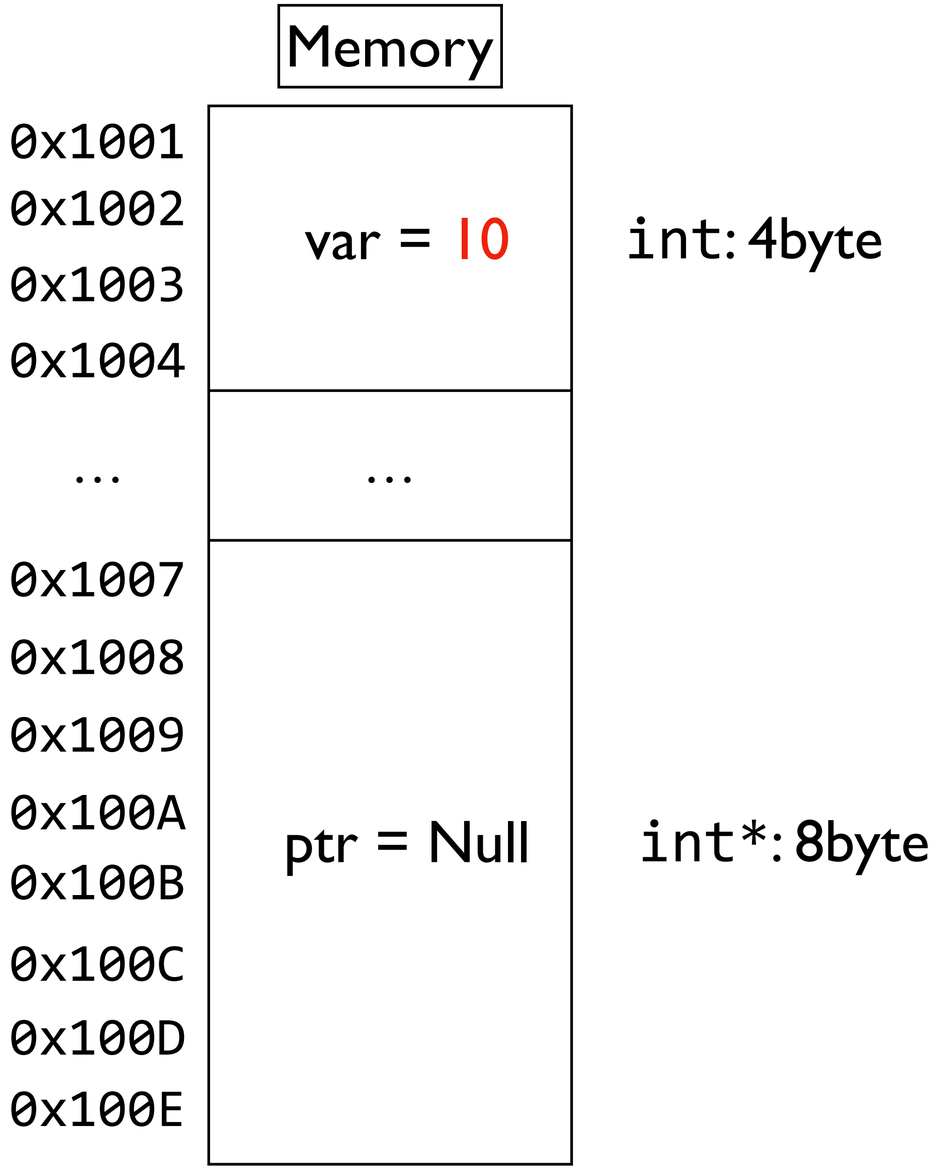
```
#include <stdio.h>
int main(){
  int var = 10;
  int *ptr;
  ptr = &var;
  printf("%d\n", *ptr);
}
```

- 포인터 변수 (Pointer variable):
  - 메모리 주소를 저장하는 변수 (예: ptr)
- 참조 (Referencing):
  - 주소 연산자 "&"를 사용하여 변수의 주소를 포인터에 할당할 수 있음 (예: ptr = &var;).
- 역참조 (Dereferencing):
  - 포인터는 포인터가 가리키는 주소에 있는 값을 검색하는 "역참조"을 지원함(예:printf("%d\n", *ptr);)

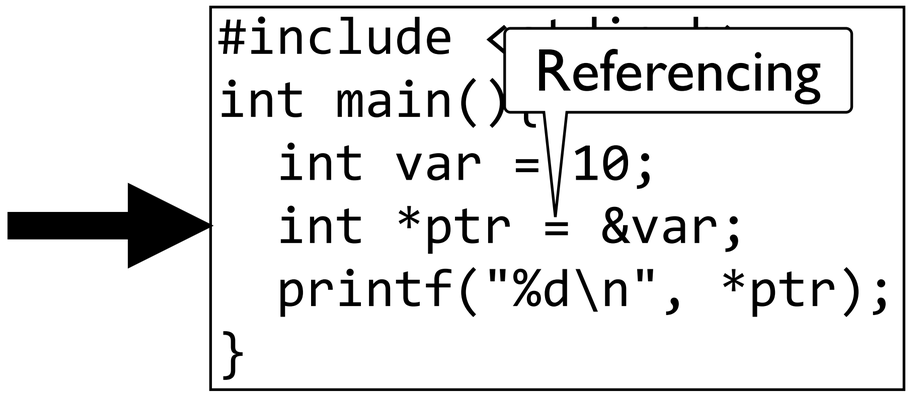
Example

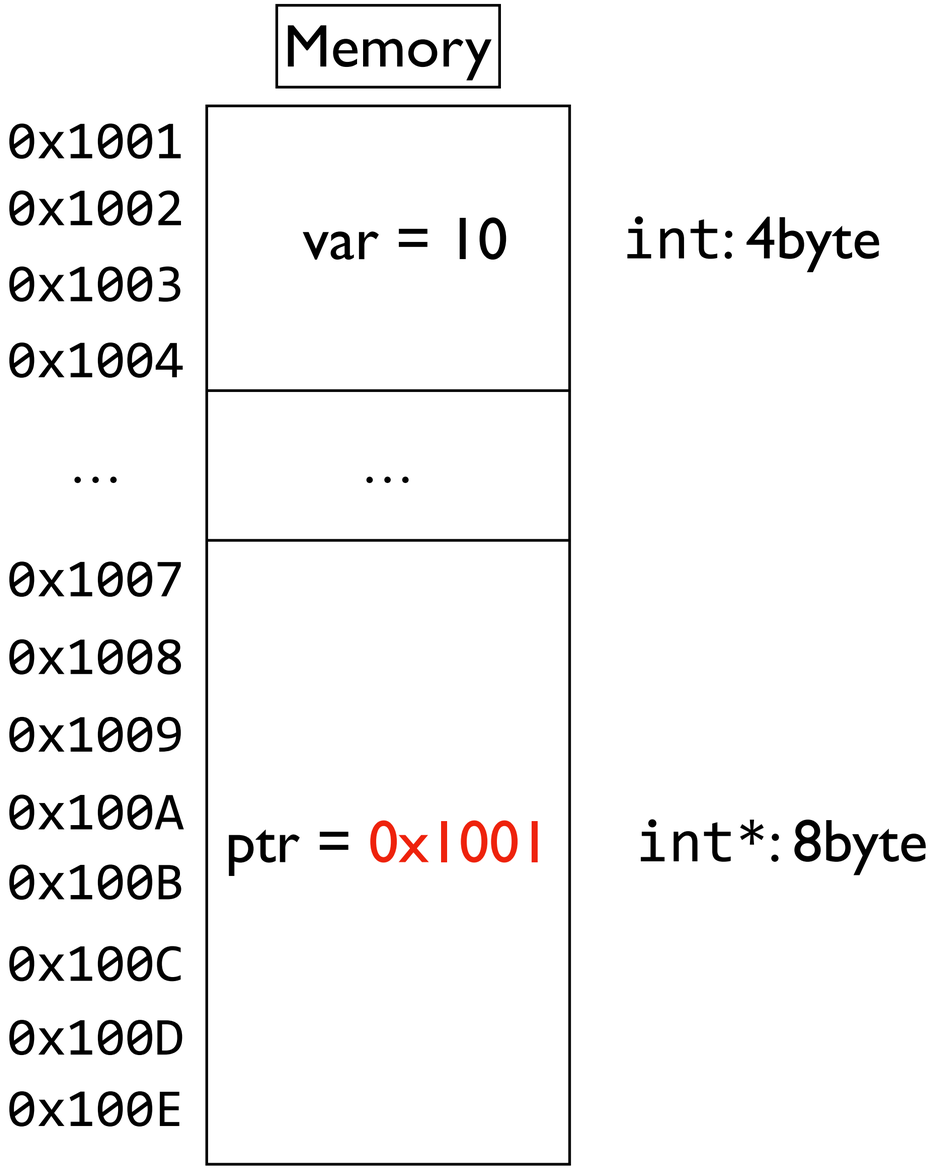


```c
#include <stdio.h>
int main(){
  int var = 10;
  int *ptr = &var;
  printf("%d\n", *ptr);
}
```



Example


#include <
int main()
int var = 10;
int *ptr = &var;
printf("%d\n", *ptr);
}
Referencing

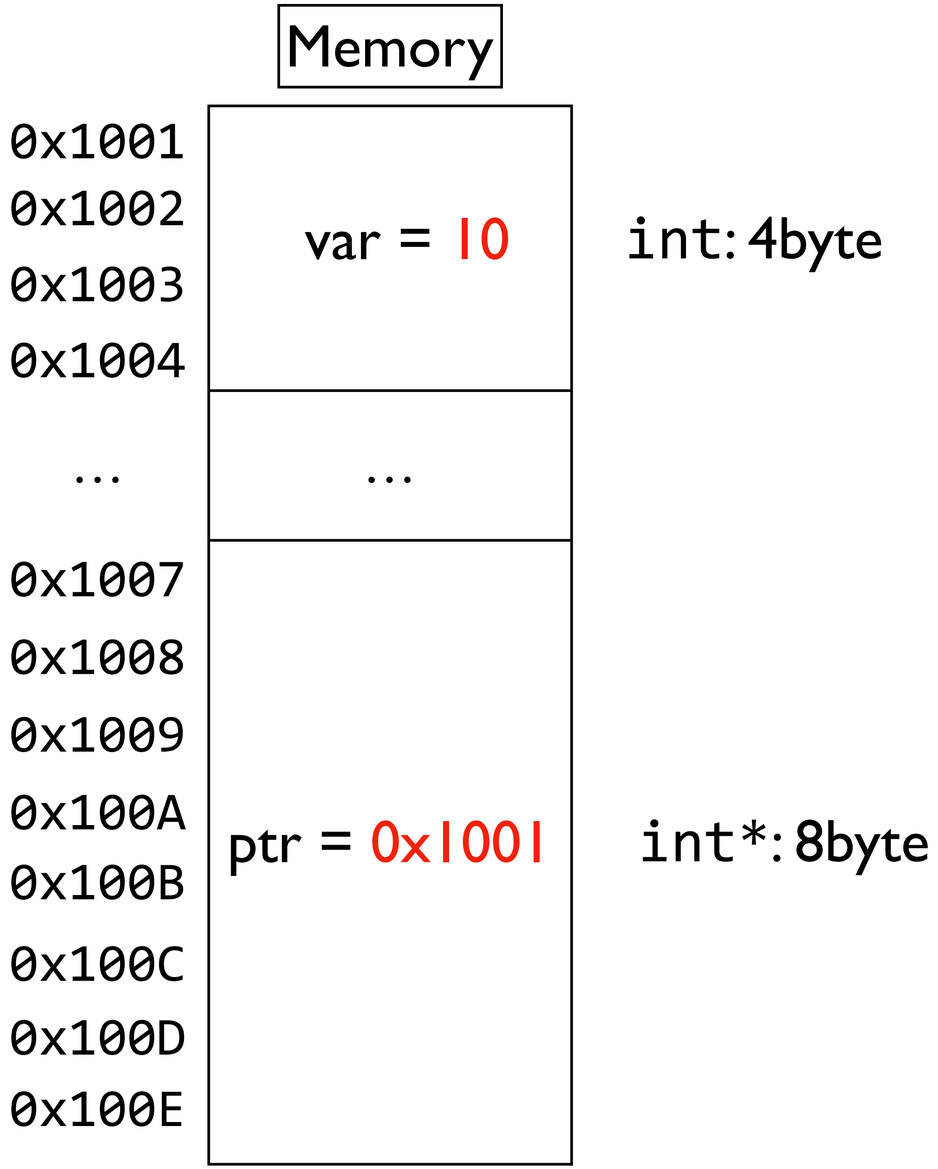

Memory
0x1001
0x1002
0x1003
0x1004
var = 10
int: 4byte
…
…
0x1007
0x1008
0x1009
0x100A
0x100B
0x100C
0x100D
0x100E
ptr = 0x1001
int*: 8byte

Example

```
#include <stdi
int main(){
  int var = 10;
  int *ptr = &var;
  printf("%d\n", *ptr);
}
```

DeReferencing

ptr변수의 값에 해당하는 주소에 들어있는 값 =10



보이드 포인터 (Void pointer)

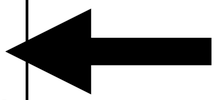
- 보이트 포인터 (void*) 는 임의의 데이터 타입을 가리킬 수 있는 특별한 (유용한) 유형의 포인터
- 단, 보이드 포인터를 역참조하려면 먼저 대상 타입으로 캐스팅해야 함.
  - Example

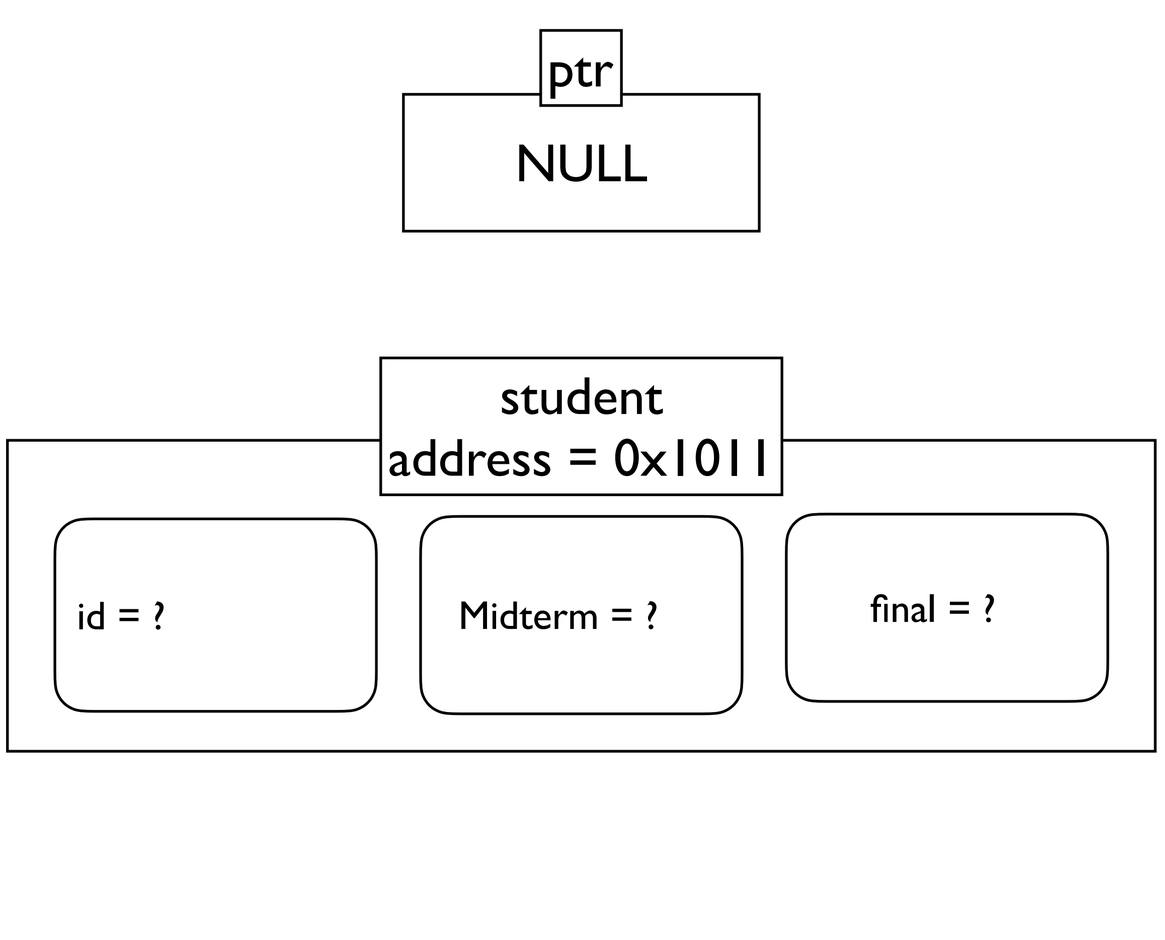
```
#include <stdio.h>

int main() {
  int x = 10;
  void *void_ptr = &x;
  int *int_ptr = (int *) void_ptr;
  printf("Value of x through void pointer: %d\n", *int_ptr);
  return 0;
}
```

C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- 포인터 변수가 구조체를 가리키는 경우 “->”를 사용하여 데이터에 접근 및 업데이트 할 수 있음

```
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  Student* ptr = &student;
  ptr->id = 2011210048;
  ptr->midterm = 60;
  ptr->final = 70;
}
```

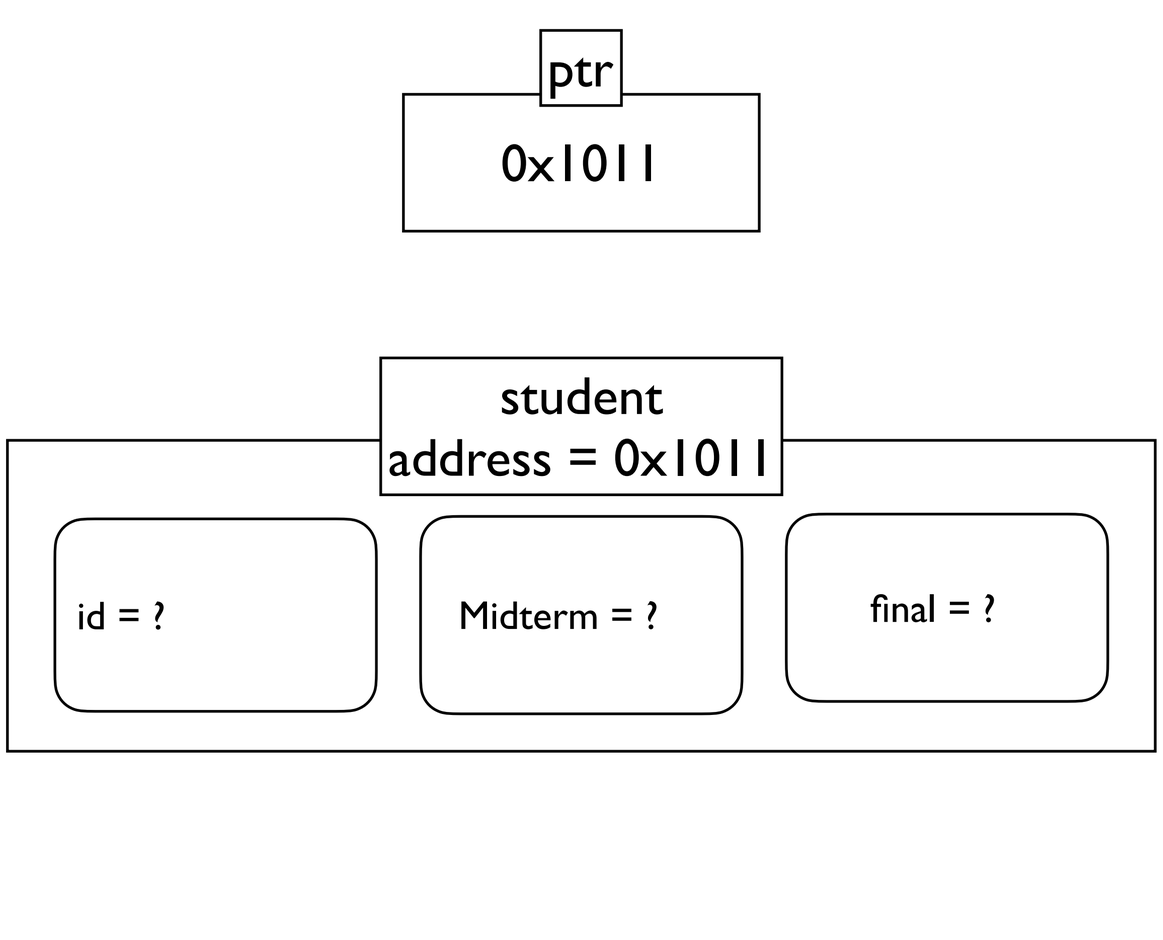




C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- 포인터 변수가 구조체를 가리키는 경우 “->”를 사용하여 데이터에 접근 및 업데이트 할 수 있음

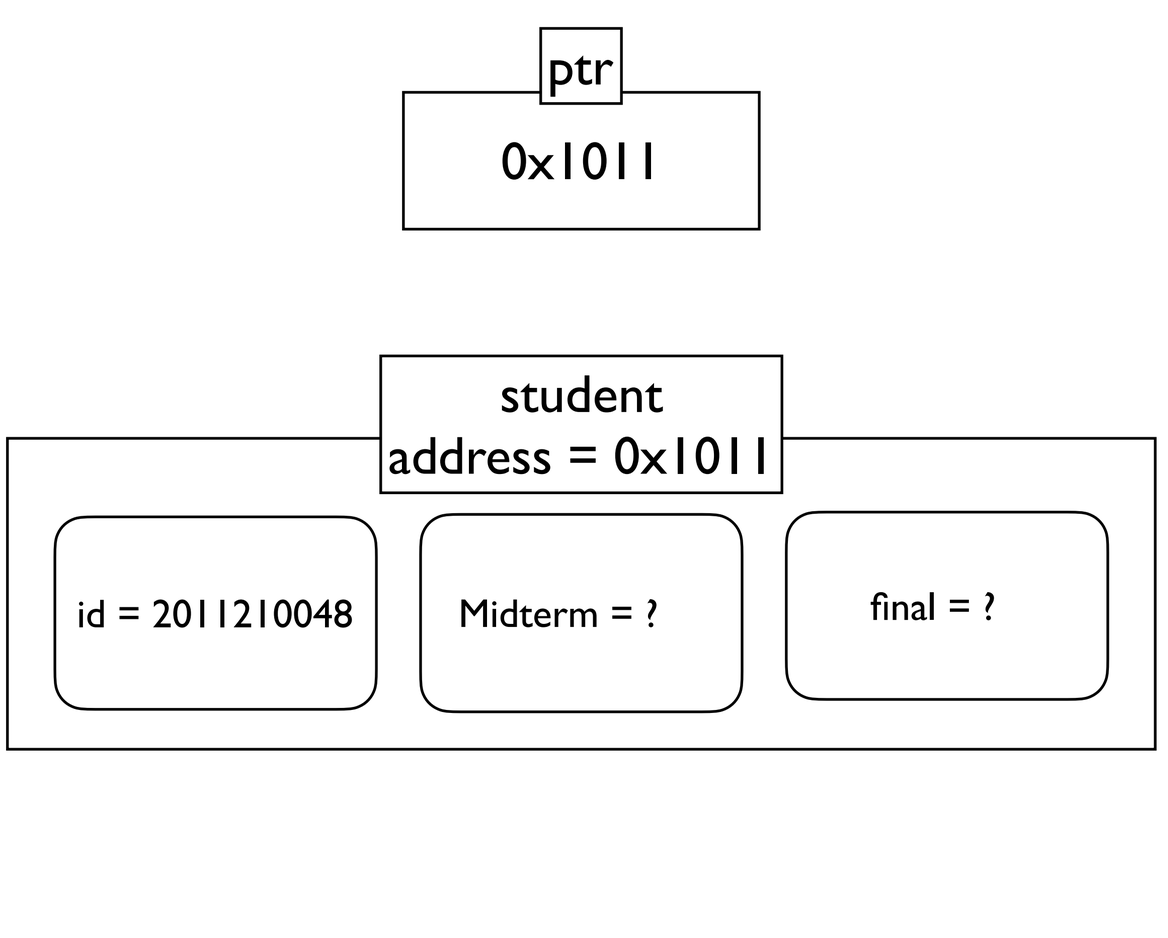
```
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  Student* ptr = &student;
  ptr->id = 2011210048;
  ptr->midterm = 60;
  ptr->final = 70;
}
```



C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- 포인터 변수가 구조체를 가리키는 경우 "->"를 사용하여 데이터에 접근 및 업데이트 할 수 있음

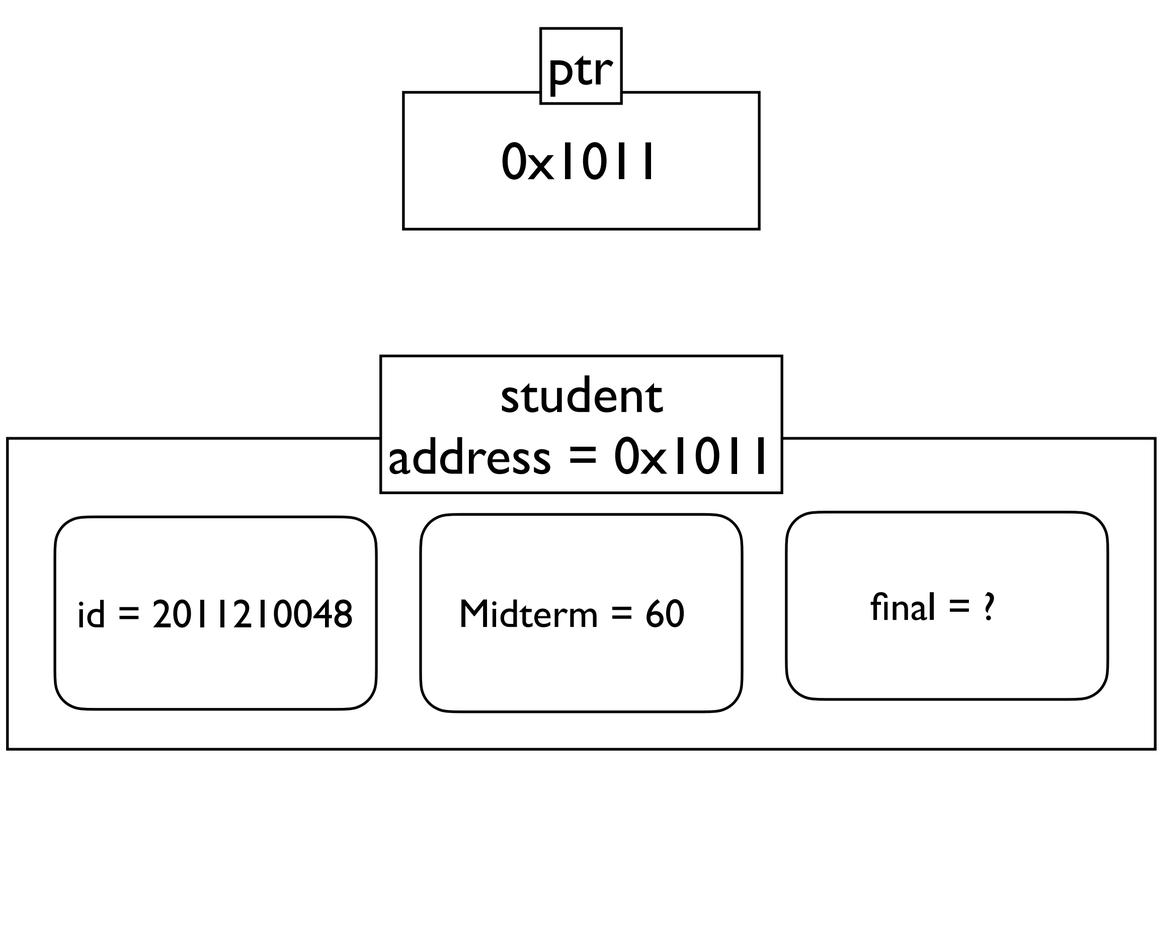
```c
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  Student* ptr = &student;
  ptr->id = 2011210048;
  ptr->midterm = 60;
  ptr->final = 70;
}
```



C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- 포인터 변수가 구조체를 가리키는 경우 “->”를 사용하여 데이터에 접근 및 업데이트 할 수 있음

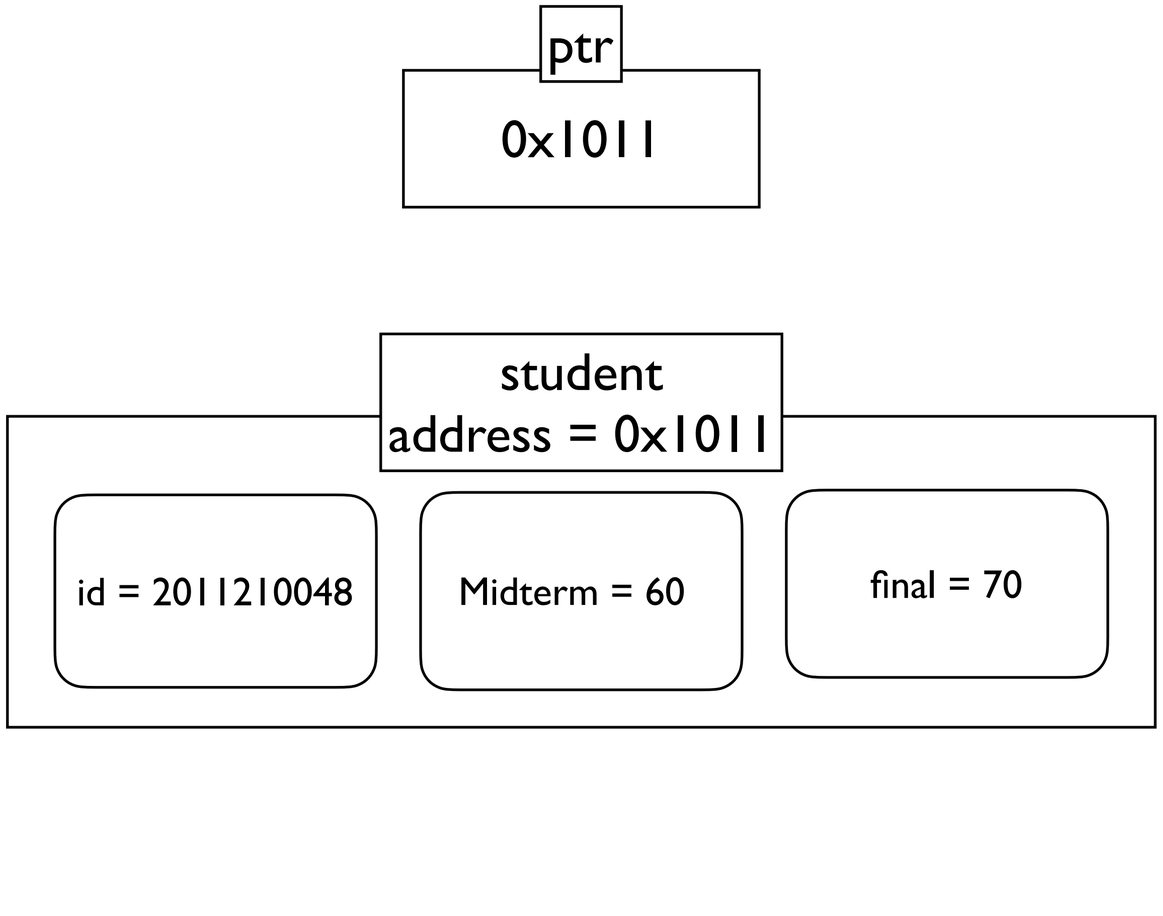
```c
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  Student* ptr = &student;
  ptr->id = 2011210048;
  ptr->midterm = 60;
  ptr->final = 70;
}
```



C 언어 리뷰 : 구조체 (Structure)

- 구조체(Structure)는 다양한 데이터 타입을 하나로 묶어 새로운 사용자 정의 데이터 타입을 만드는 방법
- 포인터 변수가 구조체를 가리키는 경우 “->”를 사용하여 데이터에 접근 및 업데이트 할 수 있음

```c
typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student student;
  Student* ptr = &student;
  ptr->id = 2011210048;
  ptr->midterm = 60;
  ptr->final = 70;
}
```



C 언어 리뷰 : 동적 메모리 할당

- C는 malloc, calloc, realloc 및 free와 같은 동적 메모리 관리를 위한 여러 기능을 제공함
  - 위 함수들은 `stdlib.h`를 통해 사용 가능함 (i.e., `#include <stdlib.h>`)
- malloc
  - 사용 목적: 특정 양의 메모리를 할당함
  - Example:

```
int* int_arr = (int *) malloc(5*sizeof(int));
Student* std_arr = (Student *) malloc(70*sizeof(Student));
```

- realloc
  - 사용 목적: 이전에 할당된 메모리 블록의 크기를 변경함.
  - Example:

```
int* ptr = (int *) malloc(5*sizeof(int));
int* ptr_ = (int *) realloc(ptr, 10*sizeof(int));
```

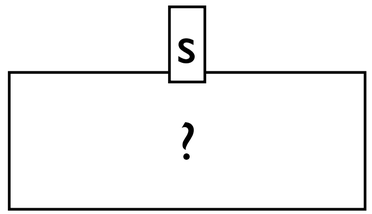
- free
  - 사용 목적: 이전에 할당된 메모리를 해제함.
  - Example:

```
int* ptr = (int *) malloc(5*sizeof(int));
free(ptr);
```

C 언어 리뷰 : 동적 메모리 할당

- C는 malloc, calloc, realloc 및 free와 같은 동적 메모리 관리를 위한 여러 기능을 제공함
- malloc: 특정 양의 메모리를 할당함

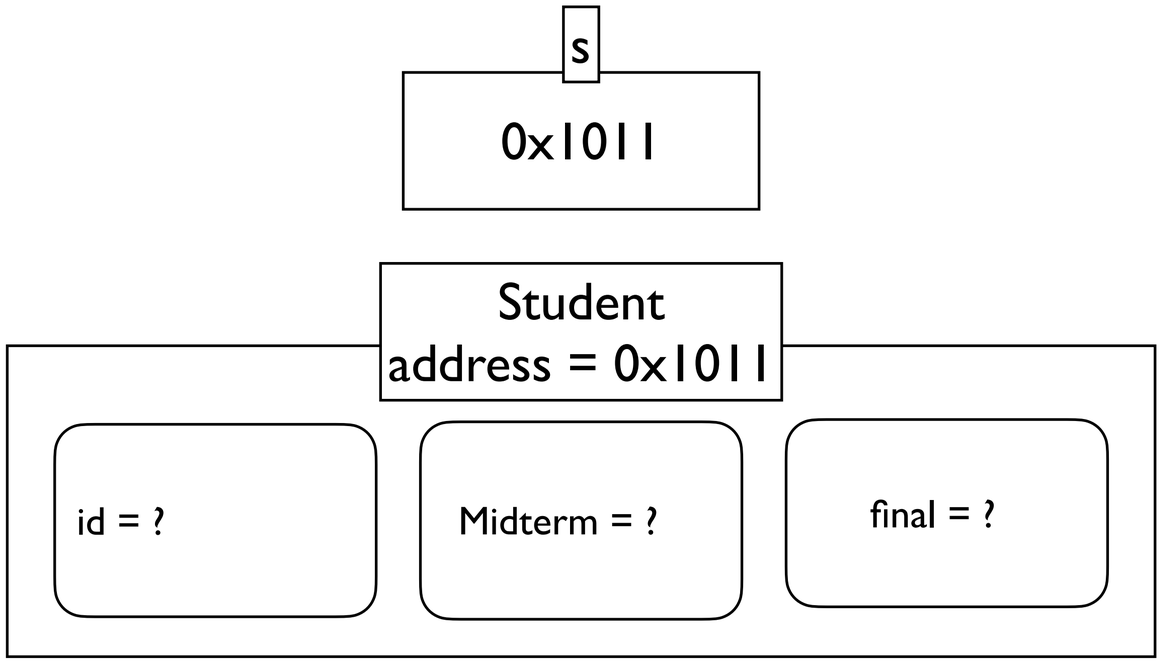
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student *s;
  s = (Student *)malloc(sizeof(Student));
  s -> id = 2011210048;
 return 0;
}
```



C 언어 리뷰 : 동적 메모리 할당

- C는 malloc, calloc, realloc 및 free와 같은 동적 메모리 관리를 위한 여러 기능을 제공함
- malloc: 특정 양의 메모리를 할당함

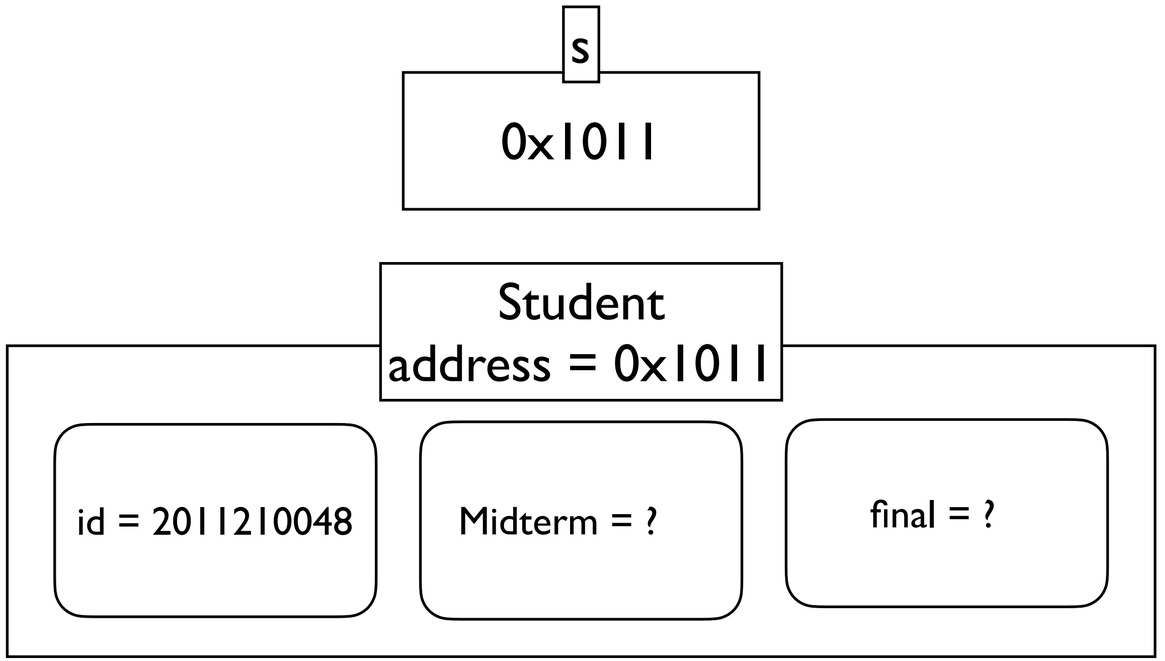
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student *s;
  s = (Student *)malloc(sizeof(Student));
  s -> id = 2011210048;
 return 0;
}
```



C 언어 리뷰 : 동적 메모리 할당

- C는 malloc, calloc, realloc 및 free와 같은 동적 메모리 관리를 위한 여러 기능을 제공함
- malloc: 특정 양의 메모리를 할당함

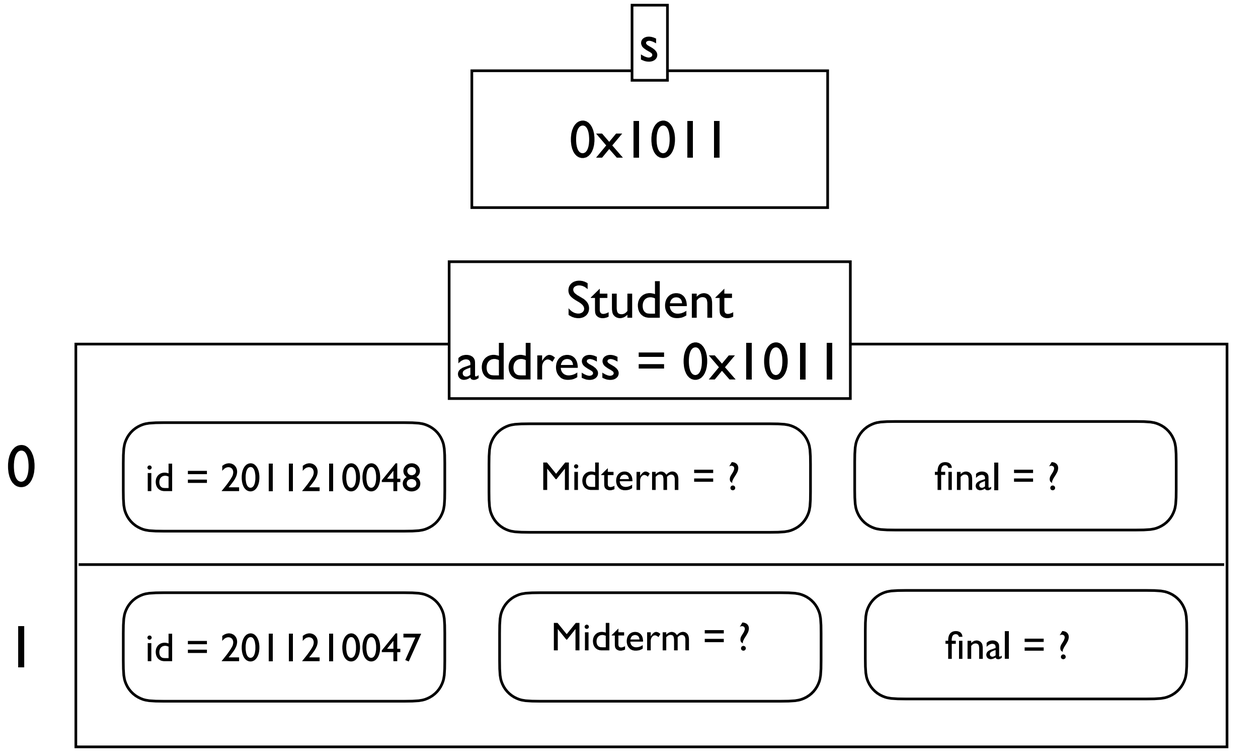
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student *s;
  s = (Student *)malloc(sizeof(Student));
  s -> id = 2011210048;
 return 0;
}
```



C 언어 리뷰 : 동적 메모리 할당

- C는 malloc, calloc, realloc 및 free와 같은 동적 메모리 관리를 위한 여러 기능을 제공함
- malloc: 특정 양의 메모리를 할당함

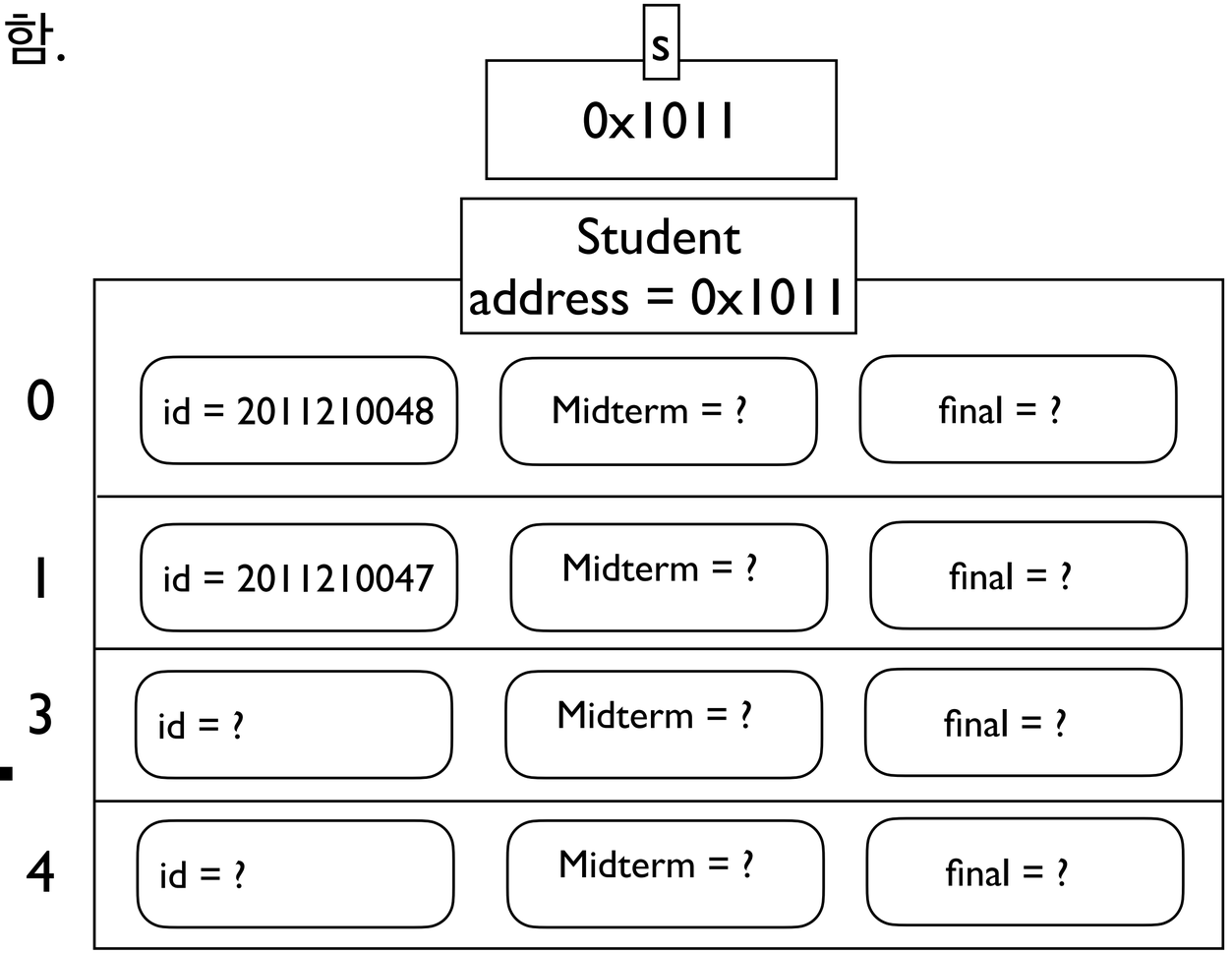
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student *s;
  s = (Student *)malloc(2 * sizeof(Student));
  s[0].id = 2011210048;
  s[1].id = 2011210047;
  return 0;
}
```



C 언어 리뷰 : 동적 메모리 할당

- C는 malloc, calloc, realloc 및 free와 같은 동적 메모리 관리를 위한 여러 기능을 제공함
- realloc: 이전에 할당된 메모리 블록의 크기를 변경함.

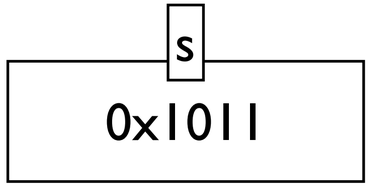
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student *s;
  s = (Student *)malloc(2 * sizeof(Student));
  s[0].id = 2011210048;
  s[1].id = 2011210047;
  s = (Student *)realloc(s, 4 * sizeof(Student));
  return 0;
}
```



C 언어 리뷰 : 동적 메모리 할당

- C는 malloc, calloc, realloc 및 free와 같은 동적 메모리 관리를 위한 여러 기능을 제공함
- free: 이전에 할당된 메모리를 해제함.

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

typedef struct {
  int id;
  int midterm;
  int final;
} Student;

int main(){
  Student *s;
  s = (Student *)malloc(2 * sizeof(Student));
  s[0].id = 2011210048;
  s[1].id = 2011210047;
  s = (Student *)realloc(s, 4 * sizeof(Student));
  free(s);
  s = NULL;
  return 0;
}
```



Example: List (리스트)

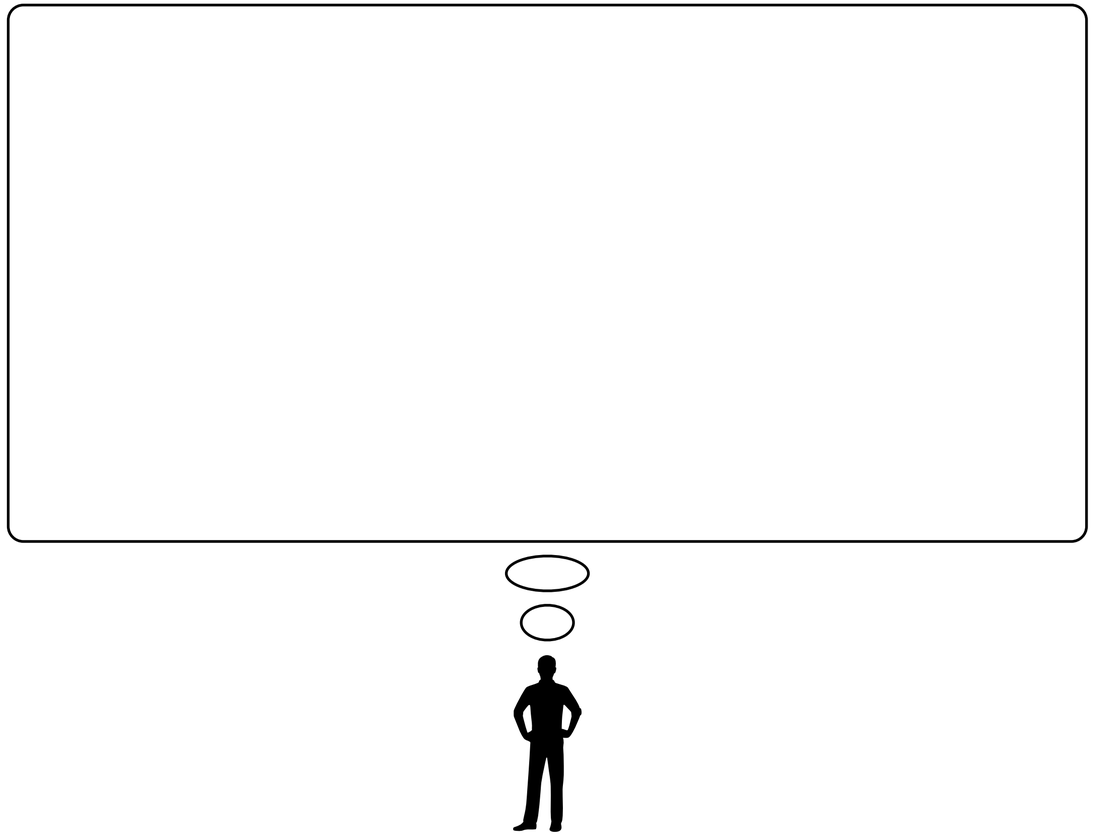
- 리스트(List)는 크기가 고정되지 않고 필요에 따라 자동으로 크기가 늘어나는 자료구조
  - 임의의 위치에 읽기 및 수정 가능

- 리스트(List.c)의 추상 자료형 (Abstract Data Type):
  - `List* create()` : 빈 리스트를 생성 후 반환
  - `int read(List *arr, int index)` : 리스트에서 주어진 인덱스에 해당하는 정수값을 반환
  - `void update(List *arr, int index, int element)` : 리스트에서 주어진 인덱스 위치에 새로운 정수 데이터를 저장
  - `void append(List *arr, int element)`: 리스트 끝에 새로운 정수를 추가함. 가득찰 경우 리스트의 용량이 늘어남.
  - `void destroy(List *arr)` : 리스트가 차지하고있는 메모리를 해제함

Example

```c
#include “List.c”
int main() {
    List *lst = create();
    for (int i = 0; i<70;++i){
      append(lst, i);
    }
    update(lst, 0, 100);
    printf(“elements:\n");
    for (int i = 0; i < 70; i++) {
        printf("%d ", read(lst, i));
    }
    destroy(lst);
    return 0;
}
```

- 사용자의 관점



배열을 이용한 리스트 구현

- 리스트는 다음과 같은 정보를 가지는 자료구조

```
typedef struct {
    int *array;
    int size;
    int capacity;
} List;
```

- create : 빈 리스트를 생성 후 반환

```
List* create() {
    List *lst = (List *) malloc(sizeof(List));
    lst->array = (int *)malloc(10 * sizeof(int));
    lst->size = 0;
    lst->capacity = 10;
    return lst;
}
```

배열을 이용한 리스트 구현

- `read` : 리스트에서 주어진 인덱스에 해당하는 정수값을 반환

```c
int read(List *lst, int index) {
    if (index >= lst->size) {
        printf("Index out of bounds\n");
        return -1; // error
    }
    return lst->array[index];
}
```

- `update` : 리스트에서 주어진 인덱스 위치에 새로운 정수 데이터를 저장

```c
void update(List *lst, int index, int element) {
    if (index >= lst->size) {
        printf("Index out of bounds\n");
        return; // error
    }
    lst->array[index] = element;
}
```

배열을 이용한 리스트 구현

- append : 리스트 끝에 새로운 정수를 추가함. 가득찰 경우 리스트의 용량이 늘어남.

```
void append(List *lst, int element) {
   if (lst->size == lst->capacity) {
        lst->capacity *= 2;
        lst->array = (int *)realloc(lst->array, lst->capacity * sizeof(int));
    }
    lst->array[lst->size] = element;
    lst->size++;
}
```

- destroy : 리스트가 차지하고 있는 메모리를 해제함

```
void destroy(List *lst) {
    free(lst->array);
    free(lst);
}
```