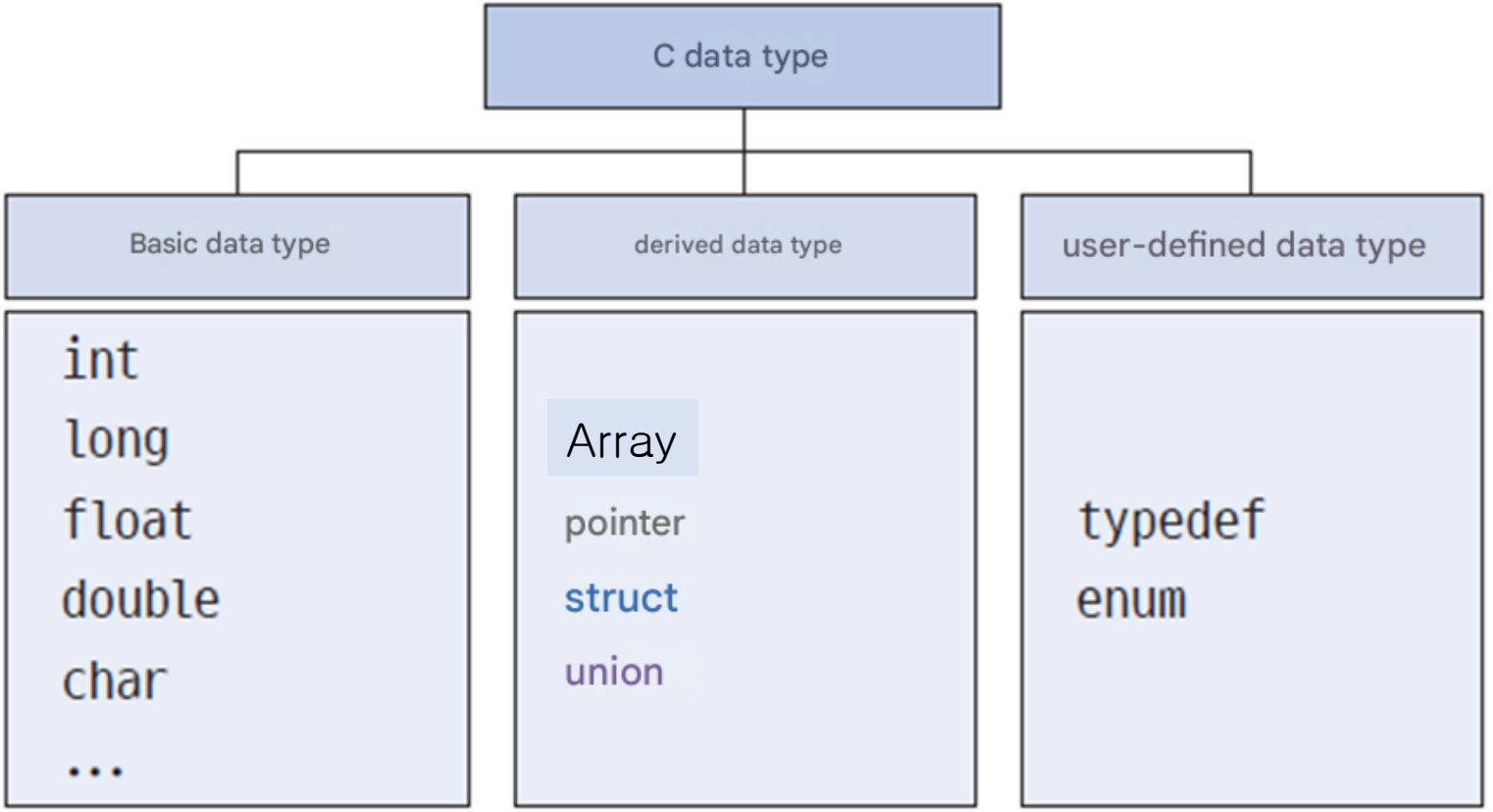
Ch.13 Structure

What you will learn in this chapter

- Structure concept, definition, and initialization method
- Relationship between structures and pointers
- Unions and typedefs

Structures are an important tool for grouping different data together .

Classification of data types



The need for structures

- How to gather data about students into one place ?

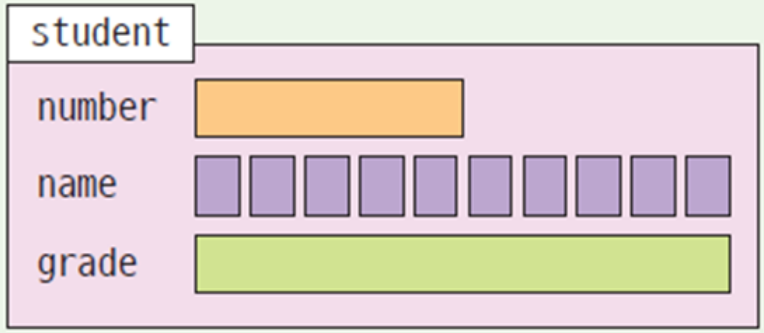
Student number: 20251234 (Number)
Name: “first last” (String)
Grade: 4.3 (Real number)
…

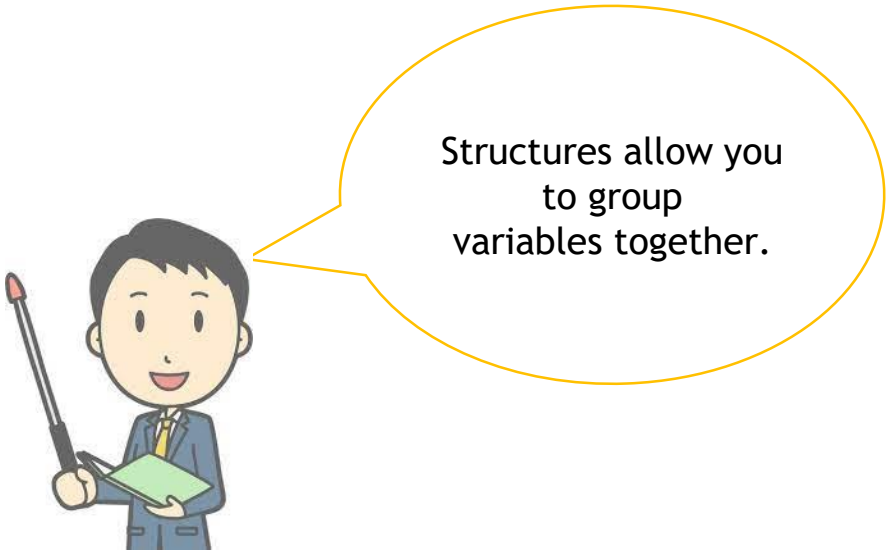
```
int number;
char name[10];
double grade;
```

It can be expressed as individual variables like this, but can it be grouped ?

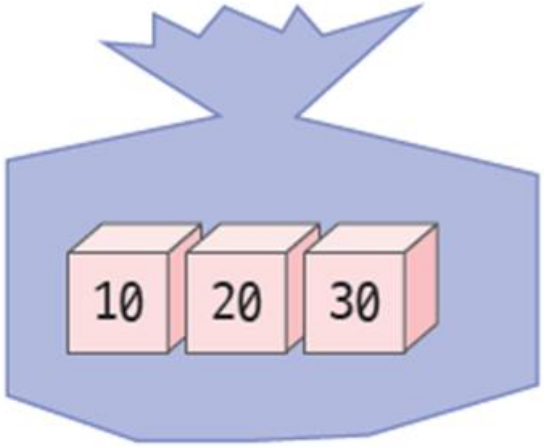
The need for structures

```
struct student {
        int number;         // student number
        char name[10];      // name
        double grade; // unit
};
```



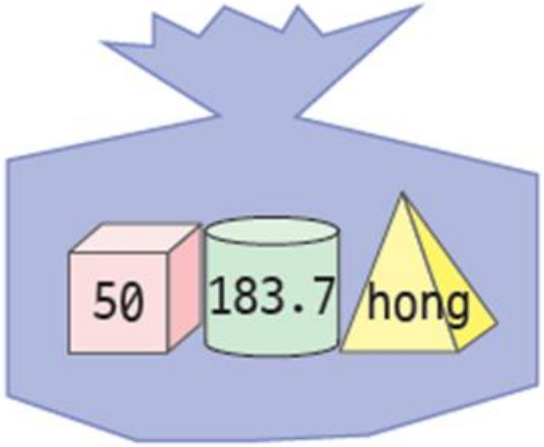


Arrays and Structures



Array

Group variables of the same type



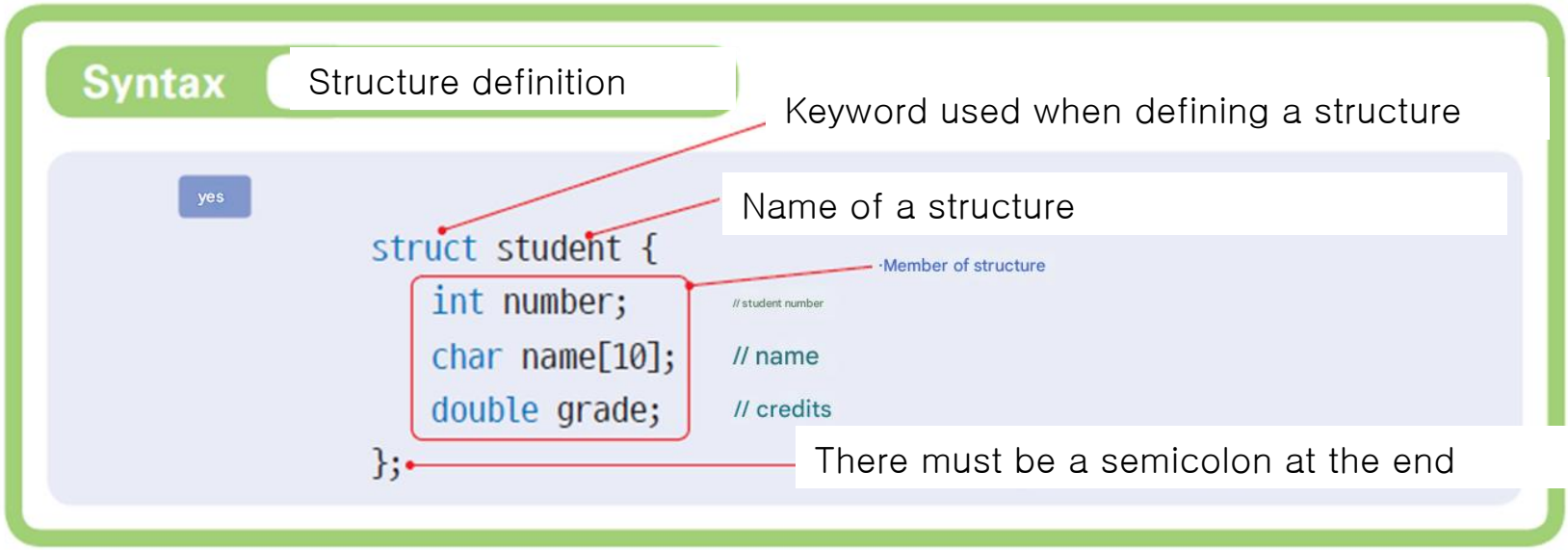
struct

Group variables of different types

Check points

1. Discuss the difference between structures and arrays .
2. Struct example
    - Student information
        . Member functions
    - Car
        . Member functions

Structure definition



A *struct definition* creates the **blueprint** (the layout) of a structure type.

struct student p1;

This line is a **declaration of a variable** of type `struct student`.

- It declares a variable `p1`.
- If `struct student` has already been **defined**, then this line also **allocates memory** for `p1`.

Structure declaration

- A structure definition is not a variable declaration.

Defining a structure is
like a mold for making waffles.

Structure
ex) int

To actually make waffles you need
to declare a structure variable.

Structure variable
ex) int a

```
struct Person {
    int age;
};              // structure definition

struct Person p1;  // variable declaration + memory allocation
```

Example of a structure definition

```
// Screen coordinates made up of x
struct point {
        int x; // x coordinate
        int y; // y coordinate
};
```

```
// date
struct date {
        int month;
        int day;
        int year;
};
```

```
// Complex number
struct complex {
        double real; // real part
        double imag ; // imaginary part
};
```

```
// square
struct rect angle {
        int x , y ;
        int width;
        int height ;
};
```

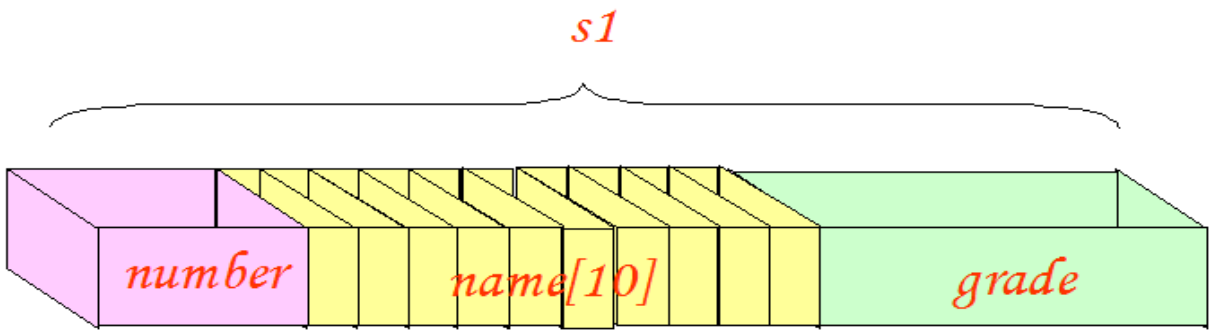
Declaring a structure variable

- Defining a structure and declaring a structure variable are different.

```
struct student {
        int number;
        char name[10];
        double grade;
};

int main(void){
        struct student s1;
        ...
}
```

Definition of Struct

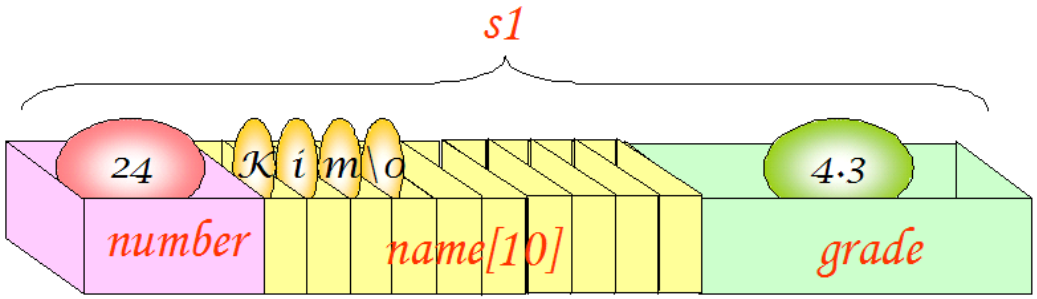
Declaration of Struct



Initializing a structure

- Initial values are listed using curly braces.

```
struct student {
        int number;
        char name[10];
        double grade;
};
struct student s1 = { 24, "Kim" , 4.3 };  // s1 is NOT a pointer,
                                          // it's variable of derived data type
```

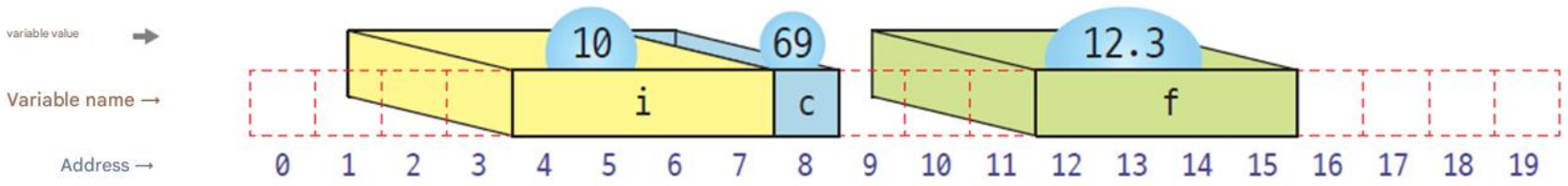


Padding happens only inside structures or arrays,
NOT between separate local variables on the stack.

char padding (struct)

Size of Struct
What is most efficient way?
Debugger

- Why does a char need 3 bytes of padding even though the smallest memory unit is 1 byte?
  - Padding exists because of **CPU alignment rules**, not because of byte-accessible memory.
- Why CPUs enforce alignment?
  - Faster access : Aligned loads/stores are much **faster**.
  - Hardware requirements : Some CPU architectures **require** aligned access for certain data types.



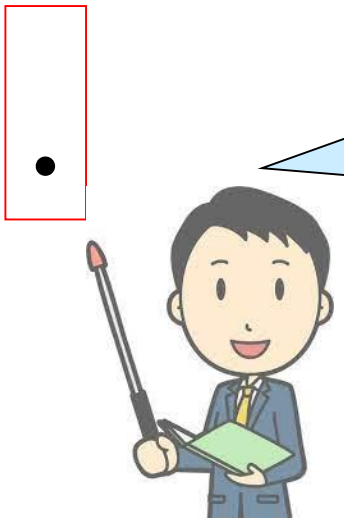
Structure member reference

Syntax Accessing structure members

struct variable — structure member

yes

```
s1.grade = 3.8;
```



. symbol is an operator used when referencing members in a structure .

Example #1

Practice
ch13-1.c

```
#include <stdio.h>
#include <string.h>
struct student {
        int number;
        char name[100];
        double grade ;
};

int main( void )
{
    struct student s;

    s.number = 20230001;
    strcpy (s.name, " Hong Gil-dong " );
    s.grade = 4.3 ;

        printf ( "Student number : %d\n" , s.number );
        printf ( "Name : %s\n" , s.name);
        printf ( “Grade : % .2f \n" , s.grade ) ;
        return 0;
}
```

Structure declaration

Declaring a structure variable

Structure member reference

```
Student number : 20230001
Name : Hong Gil-dong
GPA : 4.30
```

Example #2

```
struct student {
 int number;
 char name[10];
 double grade;
};

int main( void )
{
      struct student s;

      printf ( " Student number Enter : " );
      scanf ( "%d" , & s.number );

      printf ( " name Enter : " );
      scanf ( "%s" , s.name);

      printf ( " Credit Enter ( error ): " );
      scanf ( "%lf" , & s.grade );

      printf ( " \ nStudent number : %d\n" , s.number );
      printf ( " Name : %s\n" , s.name);
      printf ( " Grade : % .2f \n" , s.grade );
      return 0;
}
```

Structure declaration

Declaring a structure variable

Passing the address of a structure member

Debugger

```
Enter your student number : 20230001
Enter your name : Hong Gil-dong
Enter your grade ( error ): 4.3

Student number : 20230001
Name : Hong Gil-dong
GPA : 4.30
```

Initialization method

```
#include < stdio.h >
// Represents a point in two-dimensional space as a structure .
struct point {
        int x;
        int y;
};

int main( void )
{
        struct point p = { 1, 2 };
        struct point q = { .y = 2, .x = 1 };

        struct point r = p;
        r = ( struct point ) { 1, 2 };

        printf( "p=(%d, %d)\n" , p.x, p.y);
        printf( "q=(%d, %d) \n" , q.x, q.y);
        printf( "r=(%d, %d)\n" , r.x, r.y);
        return 0;
}
```

If you initialize only the first members, the rest become **zero** automatically:

// ① Values are assigned in the **order of the struct members**.

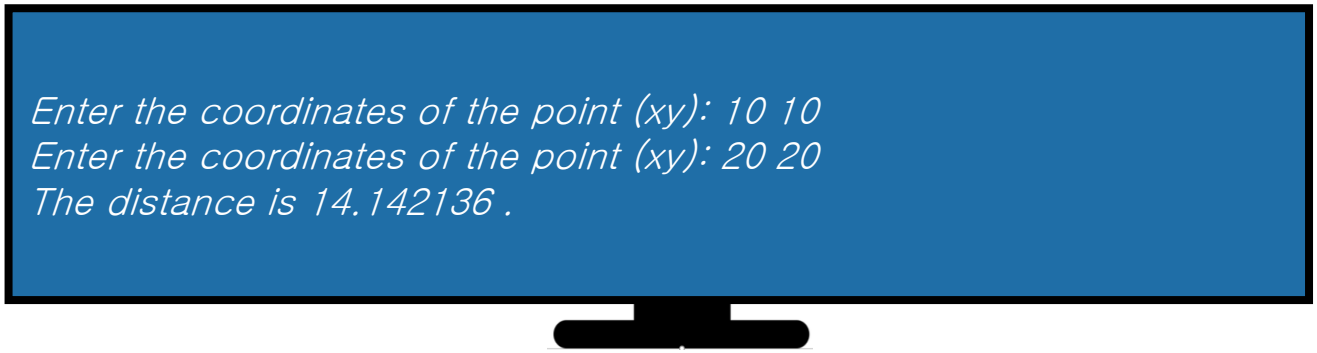
// ② Very readable, Order does not matter

// ③ structs can be copied using the = operator.

// ④ Useful when you want to "reinitialize" after the variable is declared.

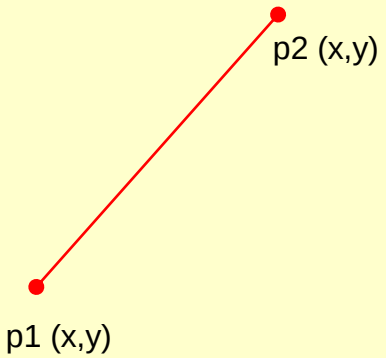
Lab: Representing points in two-dimensional space as structures

- Get the coordinates of two points from the user and calculate the distance between the two points. The coordinates of the points are expressed as a structure.



Source code

```
#include < stdio.h >
#include < math.h >
struct point {
        int x;
        int y;
};

int main( void )
{
        struct point p1, p2;
        int xdiff , ydiff ;
        double dist ;

        printf ( " dot Coordinates Enter (x y): " );
        scanf ( "%d %d" , &p1.x, &p1.y);

        printf ( " dot Coordinates Enter (x y): " );
        scanf ( "%d %d" , &p2.x, &p2.y);
```



Source code

```
        xdiff = p1.x - p2.x;
        ydiff = p1.y - p2.y;

        dist = sqrt( (double)( xdiff * xdiff + ydiff * ydiff ) );
        printf ( " The distance is % f .\n" , dist );
        return 0;
}
```

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
Enter the coordinates of the point (x y): 10 10
Enter the coordinates of the point (x y): 20 20
The distance is 14.142136 .
```

Check points

1. Each variable declared within a structure is called _____.
2. The keyword used to declare a structure is _____.
3. Why are tags needed for structures, and what is the difference between using tags and not using them?
4. Are variables created just by declaring a structure?
5. What is the operator for referencing members of a structure?

A structure that has structures as members.

```
struct date {
        int year;
        int month;
        int day;
};
```

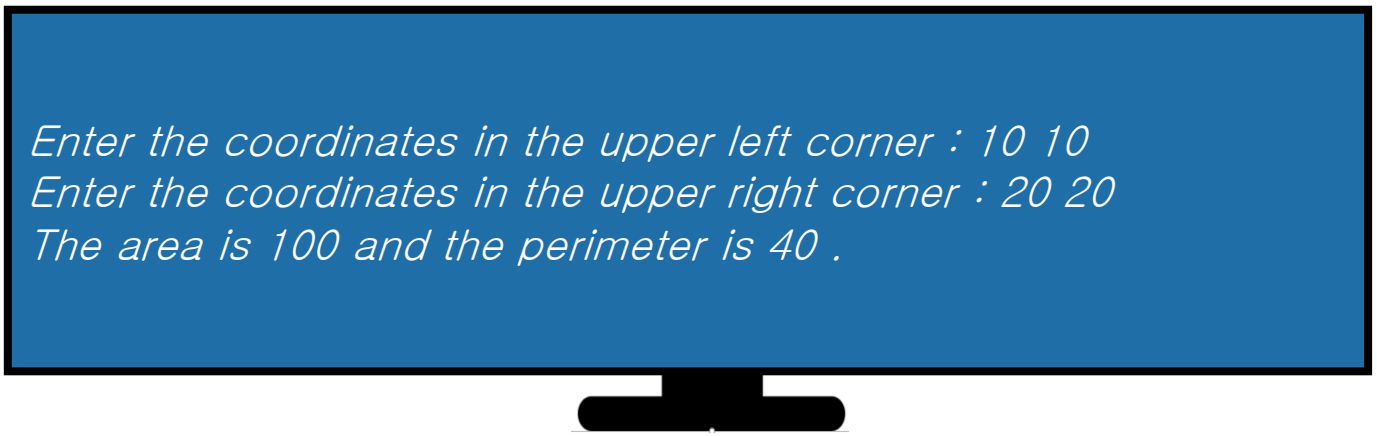
```
struct student {  // Structure declaration
        int number;
        char name[10];
        struct date d ; // structure within structure
        double grade;
};
```



```
struct student s1; // Declare structure variable

s1.d.year = 1983;  // Member reference
s1.d.month = 03;
s1.d.day = 29;
```

Lab: Representing a rectangle as a point structure

- the point structure from the previous example to represent the coordinates of the vertices .



Example

```
#include < stdio.h >

struct point {
	int x;
	int y;
};

struct rect {
	struct point p1;
	struct point p2;
};

int main( void )
{
	struct rect r;
	int w, h, area, peri;
```

Example

```
    printf( "Enter the coordinates of the upper left corner: " );
    scanf( "%d %d" , &r.p1.x, &r.p1.y);

    printf( "Enter the coordinates of the upper right corner: " );
    scanf( "%d %d" , &r.p2.x, &r.p2.y);

    w = r.p2.x - r.p1.x;
    h = r.p2.x - r.p1.x;

    area = w * h;
    peri = 2 * w + 2 * h; (둘레)
    printf( "Area is %d and perimeter is %d.\n" , area, peri);

     return 0;
}
```

Enter the coordinates in the upper left corner : 1 1
Enter the coordinates in the upper right corner : 6 6
The area is 25 and the perimeter is 20 .

Assignment and comparison of structure variables

- Assignment of variables of the same structure is possible, but comparison is not possible.

good way

```
p2 = p1;
```

Not pointer,
It's a variable of derived data type

You can do this

```
p2.x = p1.x;
p2.y = p1.y;
```

compilation error

```
if( p1 == p2 )
{
        printf("p1 and p2 are equal.")
}
```

right way

```
if( (p1.x == p2.x) && (p1.y == p2.y) )
{
        printf("p1 and p2 are equal.")
}
```

- Struct assignment is allowed because a struct is considered a single object.
  Structure assignment copies the entire memory block byte by byte.
- Array assignment is not allowed because arrays are treated as collections, not objects, and because of historical reasons and pointer-decay behavior.

Example

- Assignment of variables of the same structure is possible, but comparison is not possible .

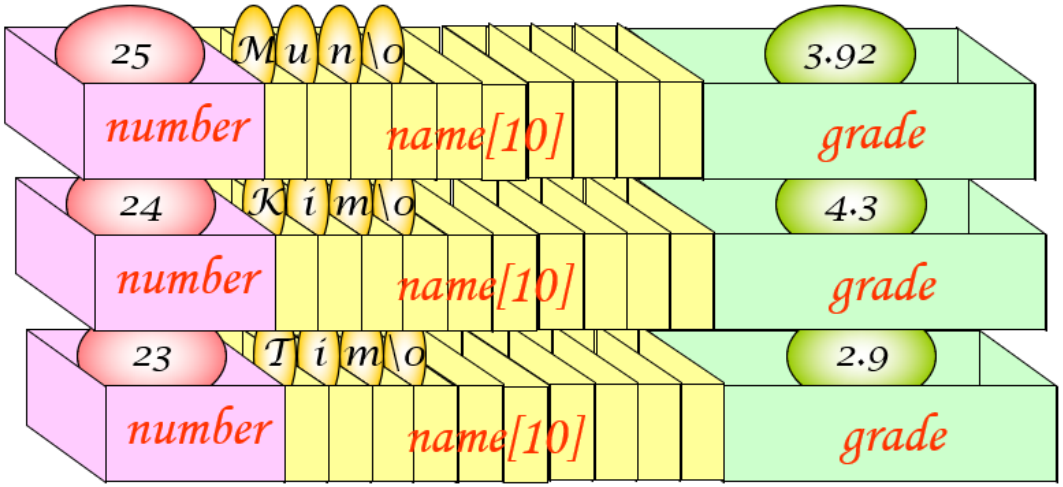
```
struct point {
	int x;
	int y;
};

int main( void )
{
	struct point p1 = {10, 20};
	struct point p2 = {30, 40};

	p2 = p1;   // Substitution possible

	if ( p1 == p2 )  // Compare -> Compile error !!
		printf ( " p1 and p2 It's the same ." )

	if ( (p1.x == p2.x) && (p1.y == p2.y) ) // Correct comparison
		printf ( " p1 and p2 It's the same ." )
}
```

Check points

1. What operations are allowed between variables of a structure ?
2. What is the difference between structure tags and structure variables ?
3. Can a structure be inserted as a structure member ?
4. Can a structure have an array as a member ?

Array of structures

- A collection of multiple structures



Array of structures

```
struct student {
        int number;
        char name[20];
        double grade;
};

int main( void )
{
        struct student list[100]; // Declare an array of structures


        list[2].number = 24;
        strcpy (list[2].name, " Hong Gil-dong " );
        list[2].grade = 4.3;
}
```

Array of structures initialization

```
struct student list[3] = {
        { 1, "Park", 3.42 },
        { 2, "Kim", 4.31 },
        { 3, "Lee", 2.98 }
};
```

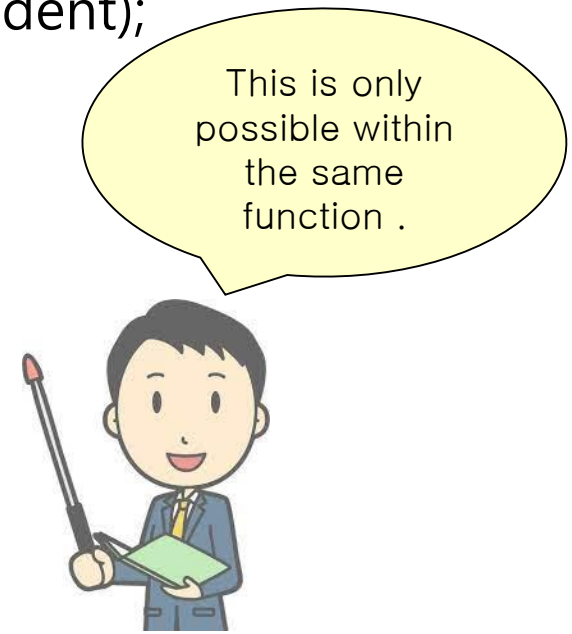
Calculate the number of elements in an array of structures

- To automatically find the number of elements in an array of structure, Divide the total number of bytes in the entire array by the number of bytes in each element .

  n = sizeof (list)/ sizeof (list[0]);

- or

  n = sizeof (list)/ sizeof (struct student);



```c
#include <stdio.h>
#define SIZE 3
struct student {
	int number;
	char name[20];
	double grade;
};

int main( void )
{
	struct student list[SIZE];
	int i;

	for ( i = 0; i < SIZE; i ++)
	{
	    printf ( " Student number Enter : " );
	    scanf ( "%d" , &list[i].number);
	    printf ( " name Enter : " );
	    scanf ( "%s" , list[i].name);
	    printf ( " Credit Enter ( error ): " );
	    scanf ( "%lf" , &list[i].grade);
	 }

	 for ( i = 0; i < SIZE; i ++)
	       printf ( " Student number : %d, Name : %s, Grade : %f\n" ,
	                   list[ i ].number, list[ i ].name, list[ i ].grade);
	 return 0;
}
```

Practice
ch13-2.c

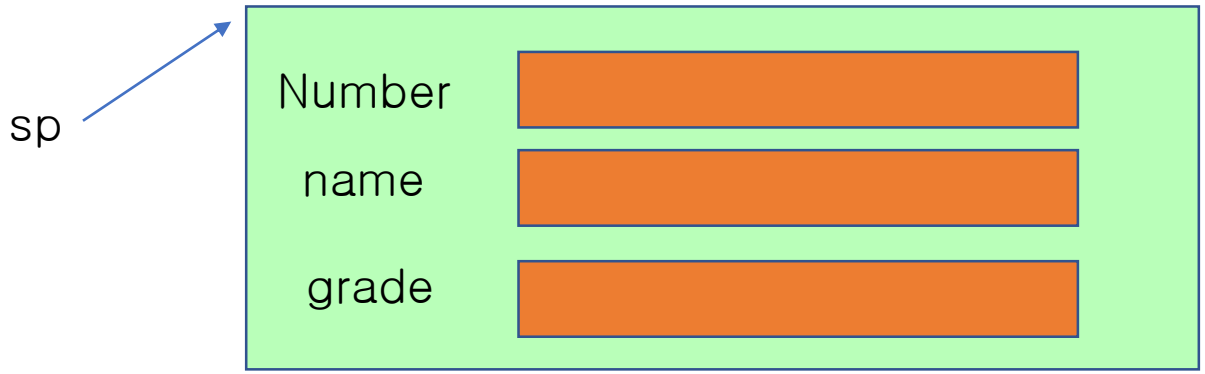
```
Enter your student number :
20190001
Enter your name : Hong Gil-dong
Enter your grade ( error ): 4.3
Enter your student number :
20190002
Enter your name : Kim Yu-shin
Enter your grade ( incorrect ): 3.92
Enter your student number :
20190003
Enter your name : Lee Seong-gye
Enter your grade ( incorrect ): 2.87
Name : Hong Gil-dong , Grade :
4.300000
Name : Kim Yu-shin , Grade :
3.920000
Name : Lee Seong-gye , Credit :
2.870000
```

Check points

1. Define an array of structures that can store information about five products. Products have number, name , and price as members .

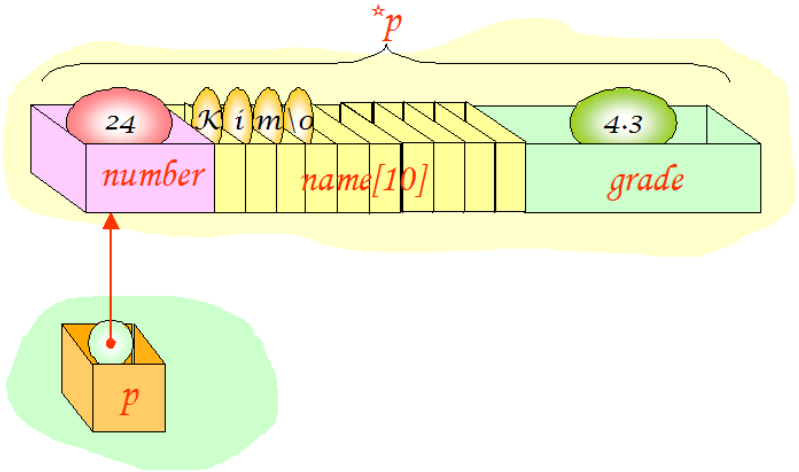
Structures and Pointers

1. Pointer to a structure
2. A structure that has pointers as members.



Pointer to a structure

```
struct student *p;

struct student s = { 24, “Kim”, 4.3 };
struct student *p;

p = &s;

printf("Student number =%d Name =%s Grade =%f \n" , s.number , s.name, s.grade );
printf("Student number =%d Name =%s Grade =%f \n" , (*p).number,(*p).name,(*p).grade);
```

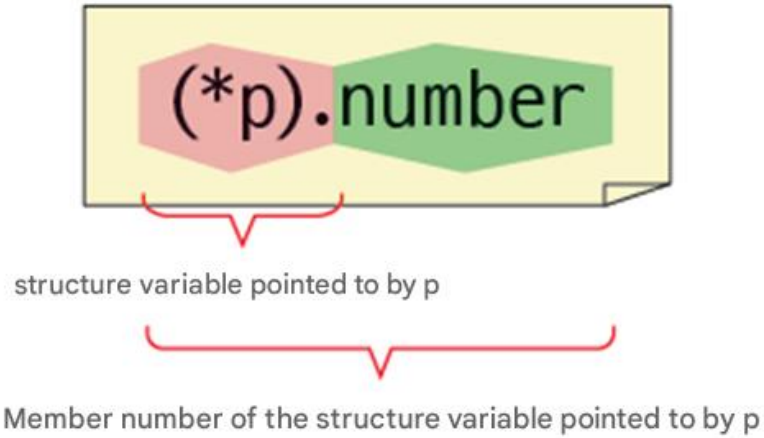


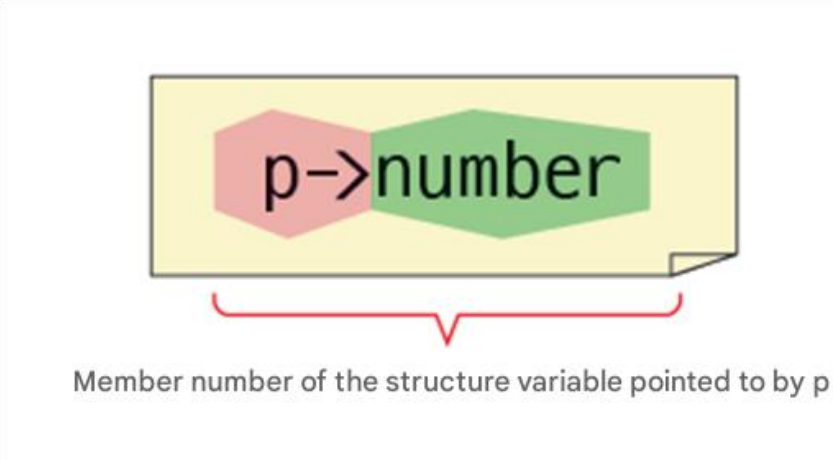
-> Operator

-> operator is used when referencing a structure member with a structure pointer.

```
struct student *p;

struct student s = { 24, “Kim”, 4.3 };
struct student *p;

p = &s;

printf("Student number =%d Name =%s Key =%f \n", p->number, p->name, p->grade);
```

-> Operator


(*p).number
structure variable pointed to by p
Member number of the structure variable pointed to by p


p->number
Member number of the structure variable pointed to by p

Example

Practice
ch13-3.c

```
// Referencing a structure through a pointer
#include <stdio.h>

struct student {
        int number;
        char name[20];
        double grade;
};

int main( void )
{
        struct student s = { 1, " Hong Gil-dong " , 4.3 };
        struct student * p;

        p = &s;

        printf("Student number=%d Name=%s Grade=%f\n" , s.number , s.name, s.grade );
        printf("Student number=%d Name=%s Grade=%f\n" , (*p).number, (*p).name, (*p).grade);
        printf("Student number=%d Name =%s Grade=%f\n" , p->number, p->name, p->grade);

        return 0;
}
```

```
Student number = 1 Name = Hong Gil-dong Grade = 4.300000
Student number = 1 Name = Hong Gil-dong Grade = 4.300000
Student number = 1 Name = Hong Gil-dong Grade = 4.300000
```

Codespaces

A structure that has pointers as members.

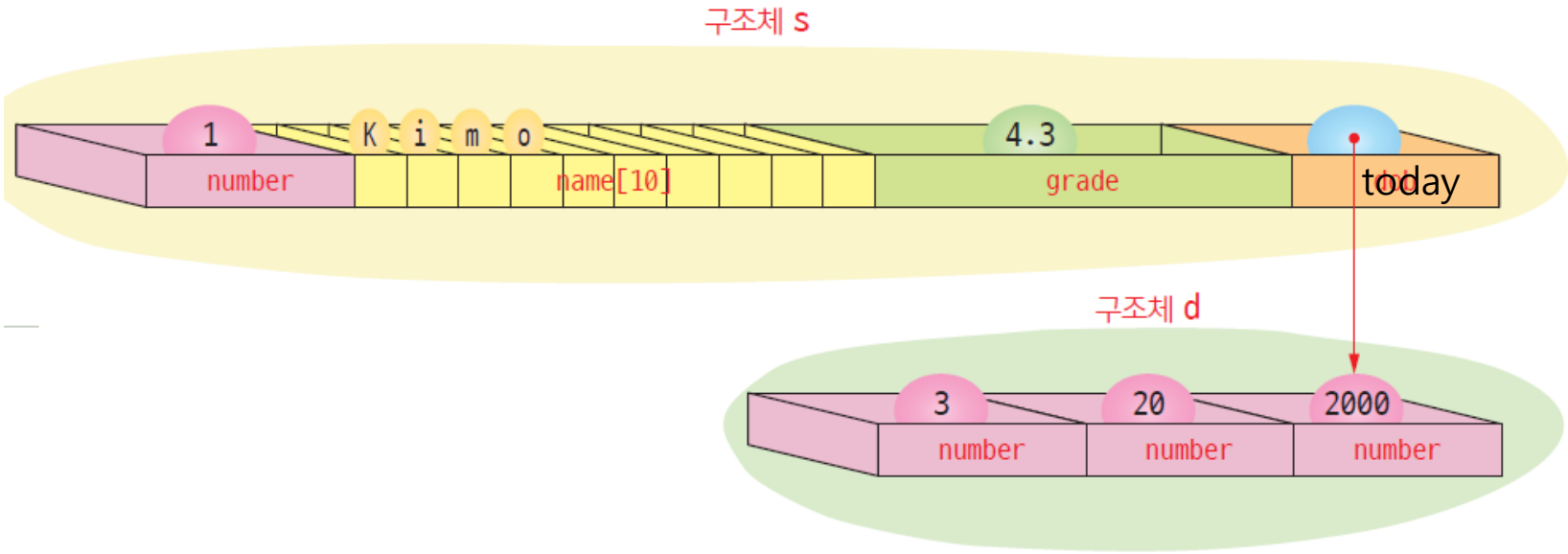
```
struct date {
        int month;
        int day;
        int year;
};

struct student {
        int number;
        char name[20];
        double grade;
        struct date *today ;
};
```

A structure that has pointers as members.

```
int main( void )
{
    struct date d = { 3, 20, 2000 };
    struct student s = { 1, "Kim" , 4.3 };

    s.today = &d;

    printf("Student number : %d\n" , s.number );
    printf("Name : %s\n" , s.name);
    printf("Grade : %f\n" , s.grade );
    printf("Birth: Year %d month %d day \n" , s.today ->year, s.today ->month, s.today ->day);

    return 0;
}
```

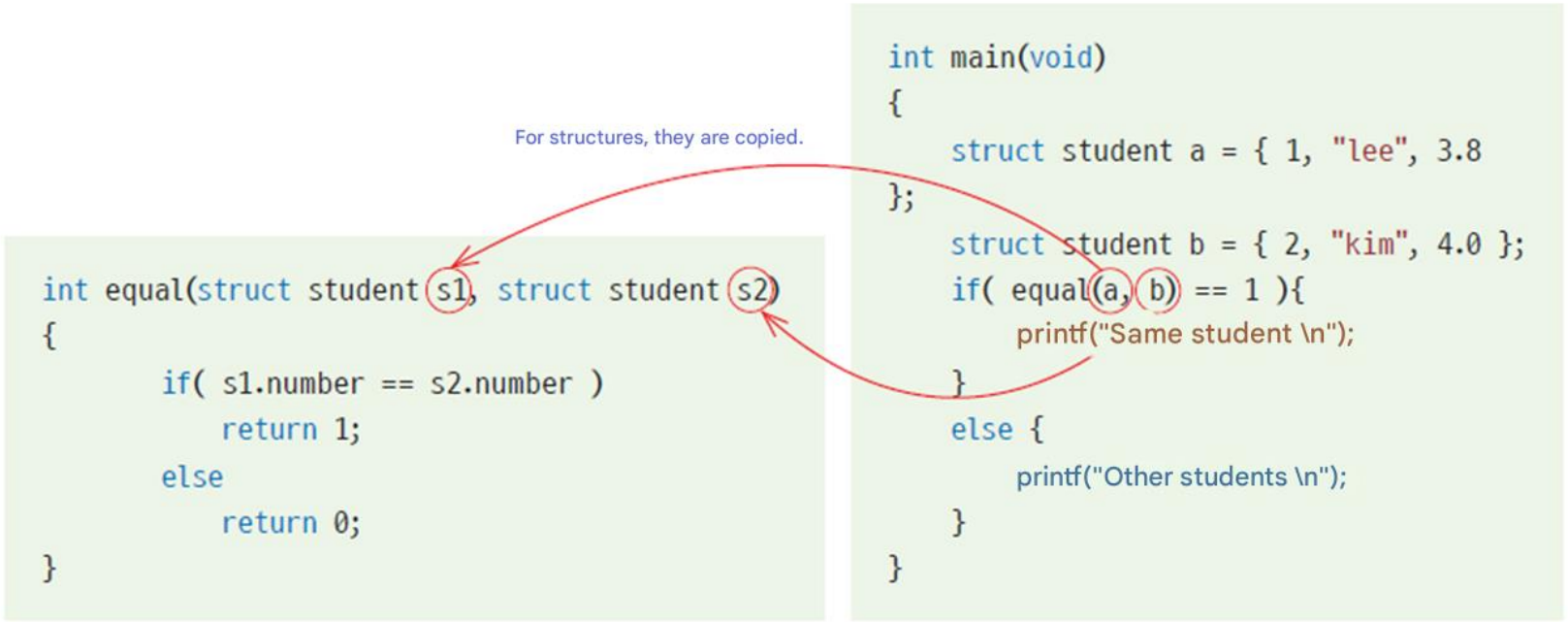
```
Student number : 1
Name : Kim
Credit : 4.300000
Birth : March 20 , 2000
```

A structure that has pointers as members.



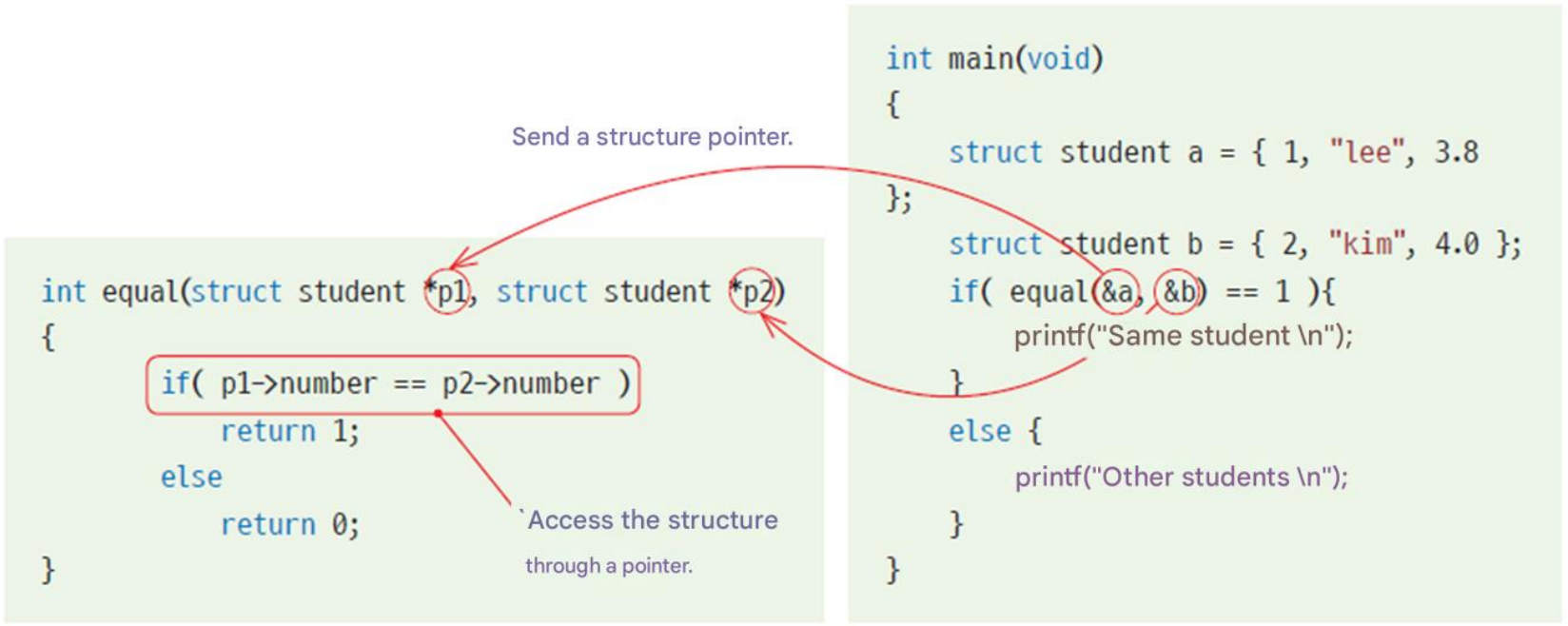
Structures and functions

- When passing *a structure as an argument to a function*
  - ***A copy* of the structure** is passed to the function .
  - If the size of the structure is large, it takes more time and memory .



Structures and functions

- When passing ***a pointer to a structure as an argument to a function***
  - It can save time and space .
  - There is a possibility that the original may be modified .



No changes via pointers

- If you only need to read the original and do not need to modify it, you can use the const keyword when defining the parameter as follows:

```
int equal(const struct student *p1, const struct student *p2)
{
        if( p1->number == p2->number )
            return 1;
        else
            return 0;
}
```

Modifying the structure through this pointer is prohibited.

Returning a structure

- A copy is returned .

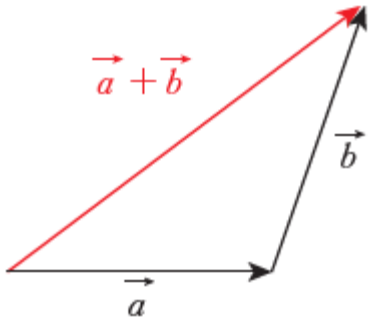
```
struct student  create()
{
        struct student s;
        s.number = 3;
        strcpy(s.name,"park");
        s.grade = 4.0;
        return s;
}
```

```
int main(void)
{
        struct student a;
        a = create();

        return 0;
}
```

Structure s is copied into structure a.

Lab: Vector Operations

- Let's create a function get_vector_sum () that calculates the sum of two vectors. This function takes two vectors as arguments, adds them, and returns the vector created as the result of the addition .



The sum of the vectors is (7.000000, 9.000000) .

Example

```
#include < stdio.h >

struct vector {
	double x;
	double y;
};
struct vector get_vector_sum ( struct vector a, struct vector b);

int main( void )
{
	struct vector a = { 2.0, 3.0 };
	struct vector b = { 5.0, 6.0 };
	struct vector sum;

	sum = get_vector_sum (a, b);
	printf ( " vector of The sum is (%f, %f) .\n" , sum.x , sum.y );

	return 0;
}
```

Example

```
struct vector get_vector_sum( struct vector a, struct vector b)
{
      struct vector result;

      result.x = ax + bx;
      result.y = ay + by;

      return result; // copy of result to caller function
}
```

The sum of the vectors is (7.000000, 9.000000) .

Check points

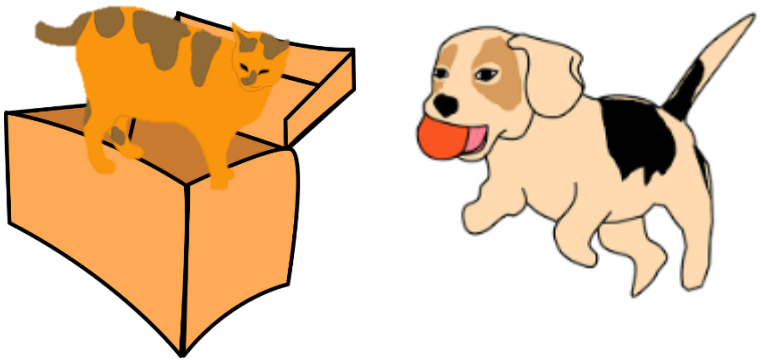
1. When passing a struct as an argument to a function, is the original passed or a copy passed?
2. When passing a pointer to a structure to a function , how can I avoid modiying the structure ?

Union

- union
  - Multiple variables share the same memory area
  - The way to declare and use a union is very similar to a structure.

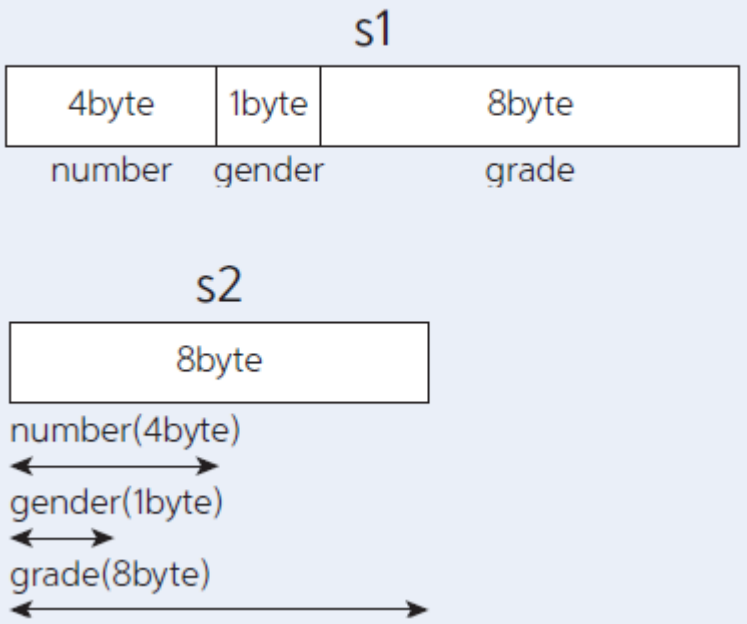
```
union example {
 char c;  // sharing the same space
 int i;     // sharing the same space
};
```

When Only One Data Type Is Used at a Time



Difference between structure and union

```
struct student
{
      int number;
      char gender;
      double grade;
} s1;

union student
{
      int number;
      char gender;
      double grade;
} s2;
```



Example

```
#include < stdio.h >

union example {
        int i ;
        char c;
};

int main( void )
{
      union example v;

      v.c = 'A' ;
      printf ( " v.c :%c v.i :% i \n" , v.c , v.i );

      v.i = 10000;
      printf ( " v.c :%c v.i :% i \n" , v.c , v.i );

}
```

Declaration of a union

Declaring a union variable .

Reference to char type .

Reference to type int .

```
vc:A vi :-858993599
vc :† vi:10000
```

Using type fields in unions

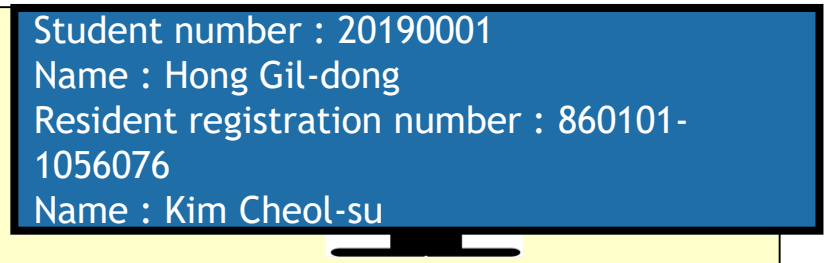
```
#include < stdio.h >
#include < string.h >
#define STU_NUMBER 1
#define REG_NUMBER 2

struct student {
        int type;
        union {
             int stu_number ;          // Student number
             char reg_number [15];   // resident registration number
        } id;
        char name[20];
};
```

Using type fields in unions

```
void print( struct student s)
{
	switch ( s.type )
	{
		case STU_NUMBER:
			printf ( " Student number %d\n" , s.id.stu_number );
			printf ( " Name : %s\n" , s.name);
			break;
		case REG_NUMBER:
			printf ( " Resident registration number : %s\n" , s.id.reg_number );
			printf ( " Name : %s\n" , s.name);
			break;
		default :
			printf ( " TypeError \n" );
			break;
	}
}
```

Using type fields in unions

```
int main( void )
{
        struct student s1, s2;

        s1.type = STU_NUMBER;
        s1.id.stu_number = 20190001;
        strcpy (s1.name, " Hong Gil-dong " );

        s2.type = REG_NUMBER;
        strcpy (s2.id.reg_number, "860101-1056076" );
        strcpy (s2.name, " Kim Cheol-su " );

        print(s1);
        print(s2);
}
```



Check points

1. The keyword used to declare a union is _____ .
2. How is the size of memory allocated to a union determined ?

Definition & Declaration of enumeration

Syntax Enum definition

An enumeration is a data type that contains a collection of symbolic constants.

yes

```
enum days { SUN, MON, TUE, WED, THU, FRI, SAT };
```

When defining an enumeration

Keywords used

Name of the enumeration

```
enum days today;
today = SUN; // OK!
```

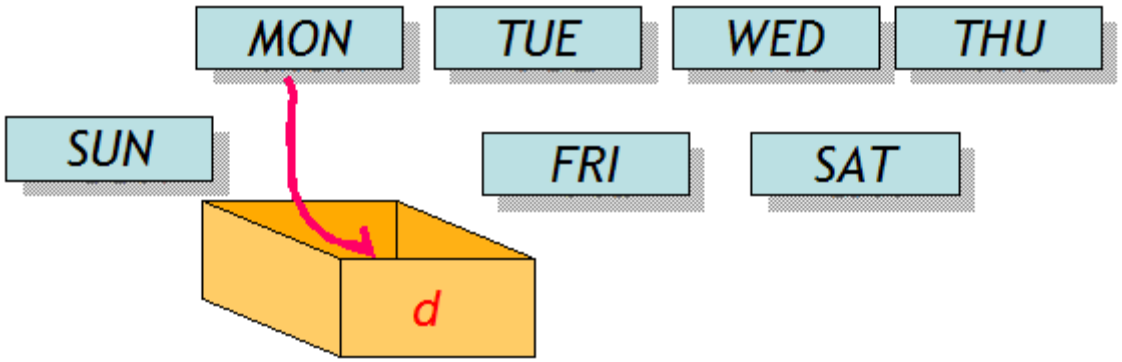
Declaring enumeration variables

variable today will be stored in memory as the integer value 0, because SUN maps to 0.
Compiler => #define SUN 0
#define MON 1
#define …. 2
int today = SUN; // becomes: int today = 0;

Enumeration

- *An enumeration is* a data type that lists in advance the values that a variable can have.
- (Example) A variable storing the day of the week can only have one of the following values :

{ Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}



Why we need enumerations

- You can write a program like this : Let's think about the problem

```
int today;
today = 0; // Sunday
today = 1; // Monday
```

- If you use enumerations,
  - It can reduce errors and improve readability.
  - The symbolic constant SUN is more preferable than 0, because its meaning is easier to understand.

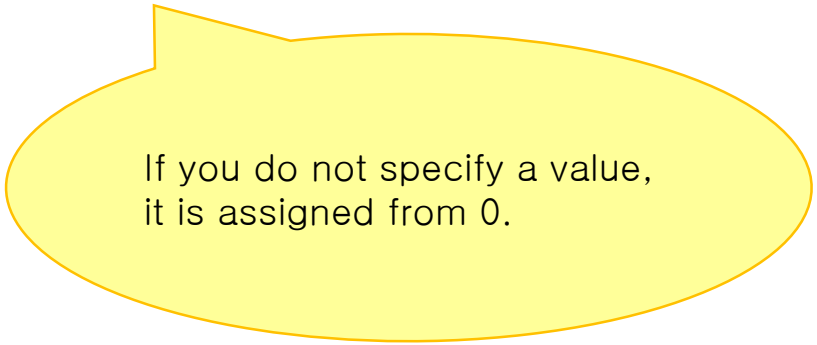
An **enum** is used when you want to represent a set of related, named integer constants. It’s helpful when a value must be one choice from a limited group — like modes, states, colors, commands, or error codes. Enums make code more readable and self-documenting, since the names carry meaning instead of using raw numbers.

Developers typically use enums when:

- A variable should hold **only one value from a fixed list**
- You want to **avoid magic numbers** scattered in your code
- You want to **improve clarity**, especially in switch statement

Enumeration Initialization

```
enum days { SUN, MON, TUE, WED, THU, FRI, SAT }; // SUN=0, MON=1, ...
enum days { SUN=1, MON, TUE, WED, THU, FRI, SAT }; // SUN=1, MON=2, ...
enum days { SUN=7, MON=1, TUE, WED, THU, FRI, SAT=6 }; // SUN=7, MON=1,
...
```



Examples of enumerations

```
enum colors { white, red, blue, green, black };
enum boolean { false, true };
enum levels { low, medium, high };
enum car_types { sedan, suv , sports_car , van, pickup, convertible };
```

Example

```
#include < stdio.h >
// Define an enum type for traffic light colors
enum TrafficLight {
     RED,    // 0
     YELLOW, // 1
     GREEN   // 2
};

int main(void) {
    enum TrafficLight light;

    light = RED;
    if (light == RED) {  printf("Stop!\n");   }

    light = GREEN;
    if (light == GREEN) {  printf("Go!\n");   }

    return 0;
}
```

Comparison of enumerations with other methods

Use of integers	Symbolic Constant	Enumeration

switch (code) {
  case 1:
  printf ("LCD TV\n");
  break ;
  case 2:
  printf (“OLED TV\n");
  break ;
}
	#define LCD 1
	#define OLED 2
	
	switch (code) {
	  case LCD:
	  printf ("LCD TV\n");
	  break ;
	  case OLED:
	  printf (“OLED TV\n");
	  break ;
	}
		enum tvtype { LCD, OLED };
		
		enum tvtype code;
		code = LCD;
		
		switch (code) {
		  case LCD:
		  printf ("LCD TV\n");
		  break ;
		  case PDP:
		  printf (“OLED TV\n");
		  break ;
		}

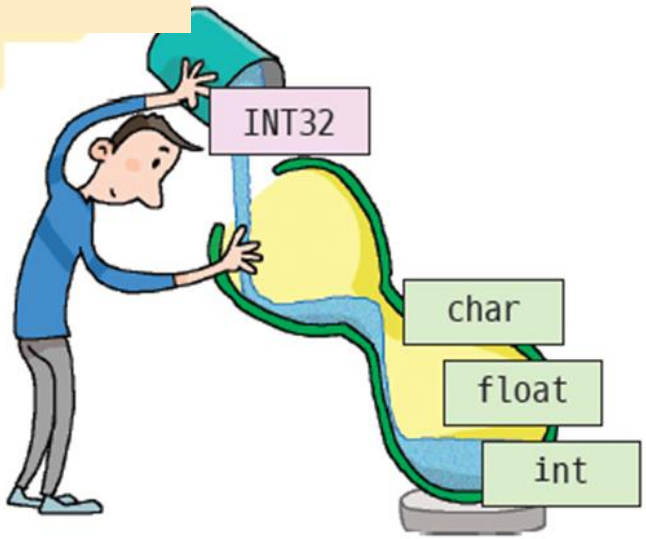
Computers are easy to understand, but people have difficulty remembering .	When writing symbolic constants.	The compiler checks to ensure that no duplication occurs .

Check points

1. The keyword used to declare an enumeration is _____ .
2. In what cases are enumerations used ?
3. If a value is not specifically specified in an enumeration, is an integer value automatically assigned ?

The concept of typedef

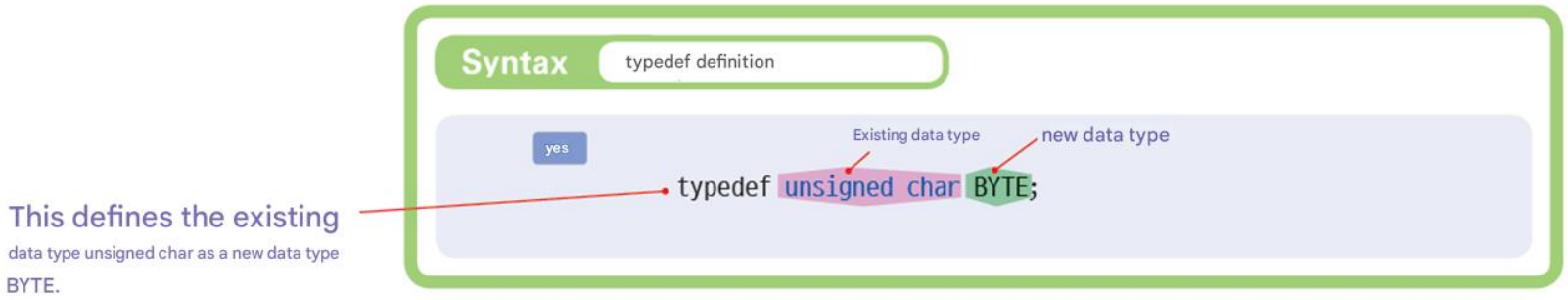
`typedef` in C is a way of giving a *new, human-friendly name* to an existing type.



We would like to announce that a new type called INT32 is now available.

typedef

`typedef` creates a new name (an alias) for an existing type.
It does *not* create a new type—just a new name for convenience.



Typedef with structure

[Without typedef]

```
struct Person {        // This defines a struct type named Person
    char name[50];
    int age;
};

struct Person s1 =  {"Nora", 27};;  // Now create a variable
```

[Without typedef]

```
struct {                    // This defines an anonymous struct (a struct with no name)
    char name[50];
    int age;
} Person;                  // It immediately declares a single variable named Person of that anonymous type.
```

```
strcpy (Person.name, "Nora”);
Person.age =  27;

Person s1;       // ERROR
```

```
struct {
    char name[50];
    int age;
} Person = {“Nora”, 27};
```

Typedef with structure

[With typedef - Recommended style]

```
typedef struct {        // no name (anonymous)
    char name[50];
    int age;
} Person;               // Not variable, now create new name (alias)

Person student;

student = {"Nora", 27};
```

[With typedef]

```
struct Person{        // struct name
    char name[50];
    int age;
};

typedef struct Person student;   // student becomes a typedef alias for struct Person

student s = {"Nora", 27};
```

Example of typedef

```
typedef   unsigned char BYTE;
BYTE index; // Same as unsigned int index;

typedef int INT32;
typedef unsigned int UINT32;

INT32 i; // Same as int
UINT32 k; // Same as unsigned int k ;
```

Defining a new type as a struct

- As a structure You can define new types .

```
struct point {
    int x;
    int y;
};
typedef struct point POINT ;
POINT a, b;
```

```
typedef struct complex {
    double real;
    double image;
} COMPLEX;
COMPLEX x, y;
```

Check points

1. What is the use of typedef ?
2. What are the advantages of typedef ?
3. Let 's define a structure representing an employee and define it as a new type called employee using typedef .

Q & A