

Ch.13 Structure

What you will learn in this chapter

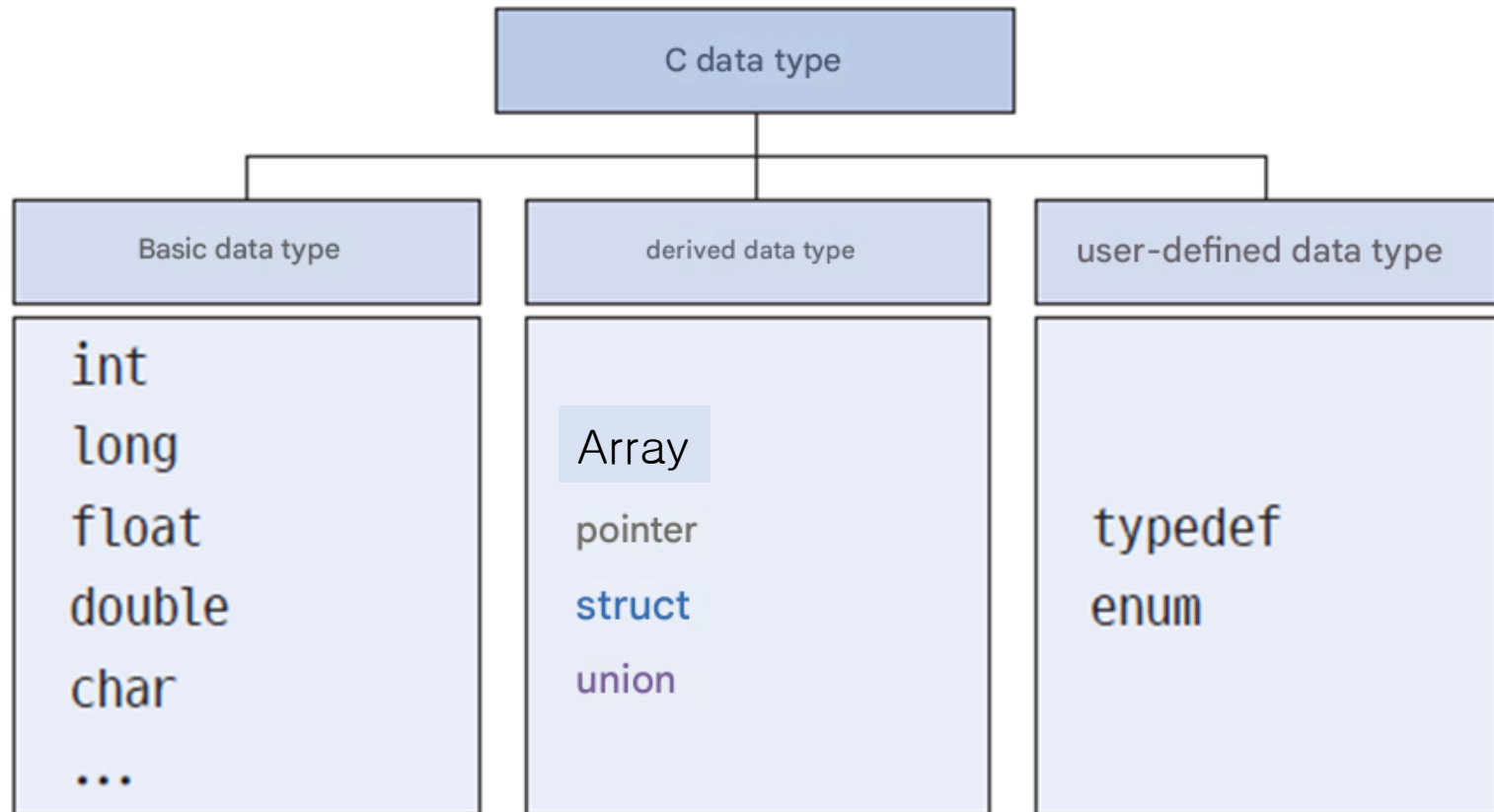


- Structure concept, definition, and initialization method
- Relationship between structures and pointers
- Unions and typedefs

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)
...

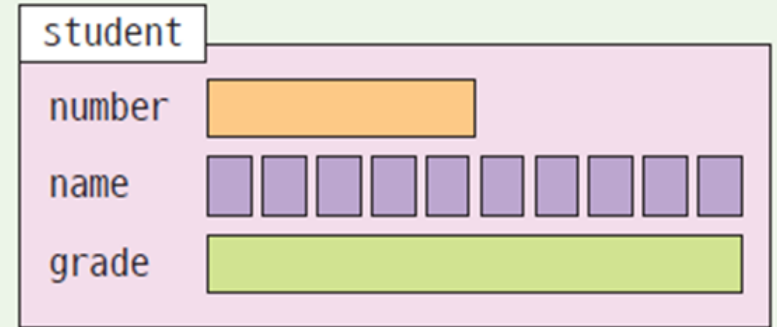


```
int number;  
char name[10];  
double grade;
```

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

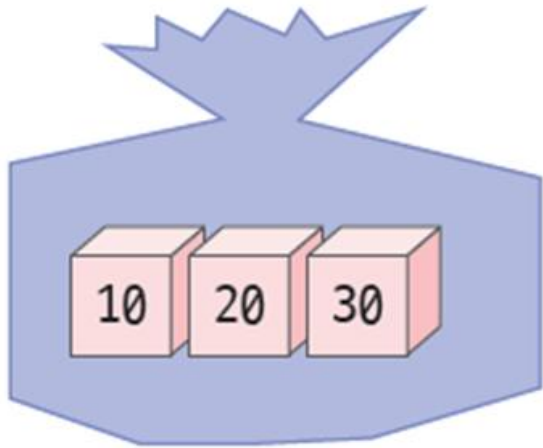
The need for structures

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



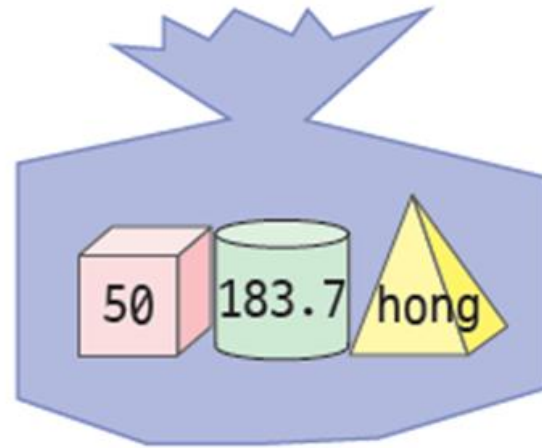
Structures allow you
to group
variables together.

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 declaration

Syntax

struct declaration

yes

```
struct student {
```

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

```
};
```

Keywords used when defining a structure

·Name of the structure (tag)

·Member of structure

// student number

// name

// credits

There must be a semicolon at the end.

Structure declaration

- A structure declaration is not a variable declaration.

Defining a structure is like a mold for making waffles.



structure

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



struct variable

Example of a structure declaration

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

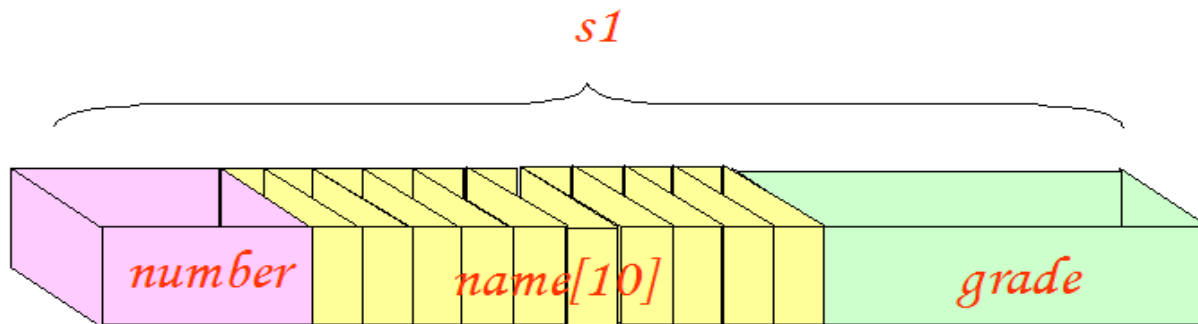
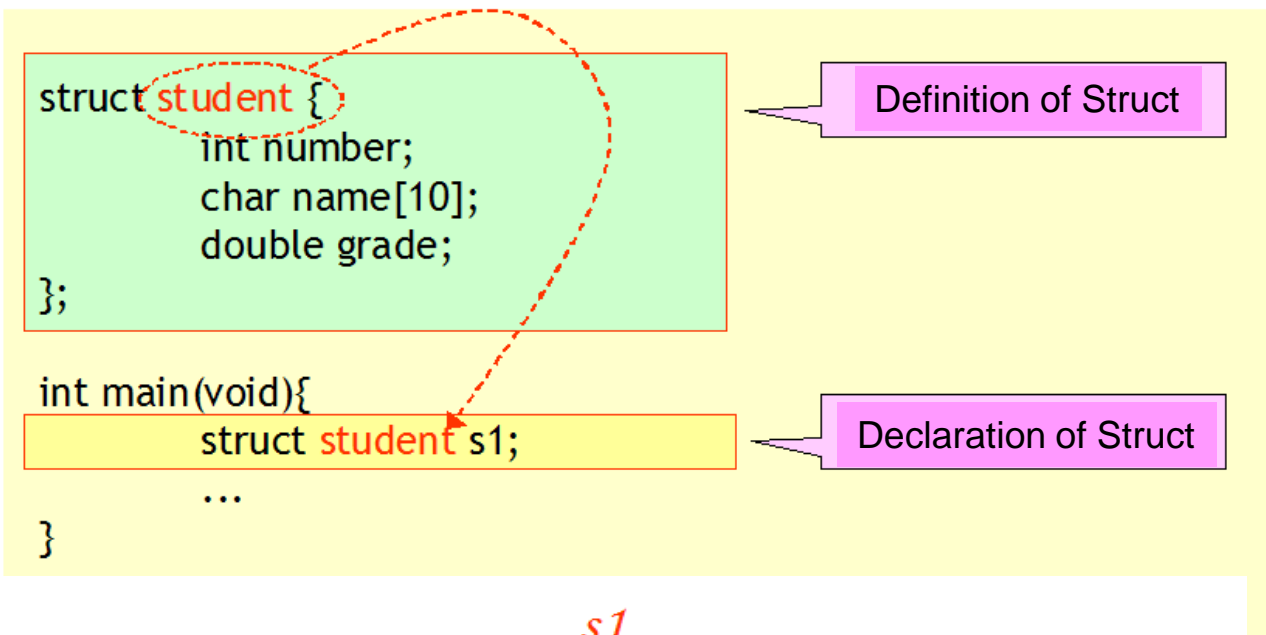
```
// date
struct date {
    int month;
    int day;
    int year;
};
```

```
// 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

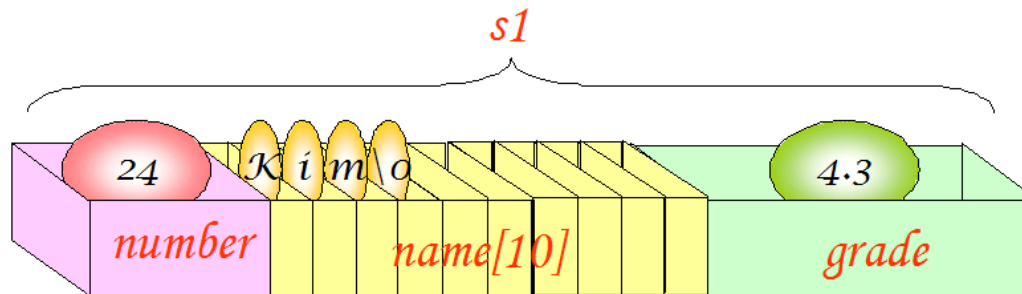
- Defining a structure and declaring a structure variable are different.



Initializing a structure

- Initial values are listed using curly brackets .

```
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
```



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

Codespace

Practice

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

```
#include <string.h>
```

```
struct student {  
    int number;  
    char name[100];  
    double grade ;  
};
```

Structure
declaration

```
int main( void )
```

```
{
```

```
    struct student s;
```

Declaring a
structure
variable

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

Structure
member
reference

```
    printf ( "Student number : %d\n" , s.number );
```

```
    printf ( "Name : %s\n" , s.name);
```

```
    printf ( "Grade : %.2f \n" , s.grade ) ;
```

```
    return 0;
```

```
}
```

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

Example #2

Codespace

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

Structure
declaration

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

```
    struct student s;
```

Declaring a structure
variable

```
    printf ( " Student number Enter : " );
```

```
    scanf ( "%d" , & s.number );
```

Passing the address of
a structure member

```
    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;
```

```
}
```

```
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
```

New initialization method

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

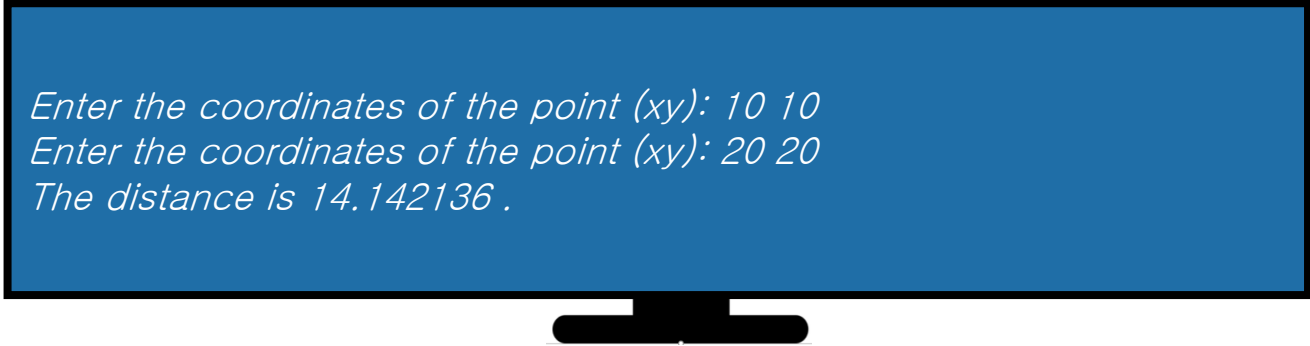
int main( void )
{
    struct point p = { 1, 2 };           // ①
    struct point q = { .y = 2, .x = 1 }; // ②

    struct point r = p;                   // ③
    r = ( struct point ) { 1, 2 };        // ④

    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;
}
```


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 .



*Enter the coordinates of the point (xy): 10 10
Enter the coordinates of the point (xy): 20 20
The distance is 14.142136 .*

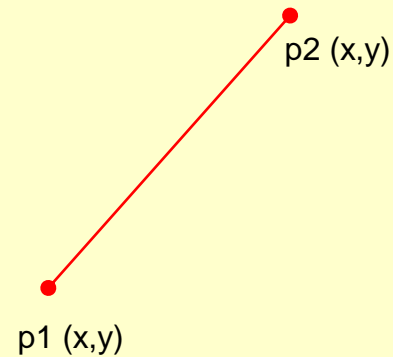
Source code

```
#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;  
}
```

```
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 dob ; // structure within structure  
    double grade;  
};
```

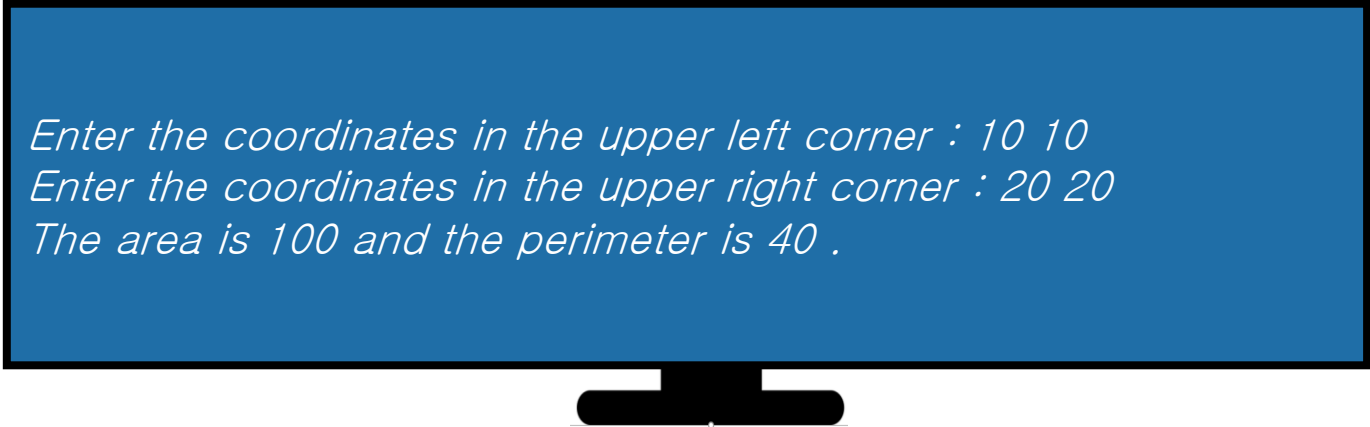


```
struct student s1; // Declare structure variable
```

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

Lab: Representing a rectangle as a point structure

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

A computer monitor with a black frame and a black stand. The screen is blue and displays three lines of white text.

Enter the coordinates in the upper left corner : 10 10
Enter the coordinates in the upper right corner : 20 20
The area is 100 and the perimeter is 40 .

Example

Codespace

```
#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.y - r.p1.y;
```

```
area = w * h;  
peri = 2 * w + 2 * h;  
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 .

`p2 = p1;` good way

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



`p2.x = p1.x;`
`p2.y = p1.y;` You can do this



`if(p1 == p2)` compilation error

{

`printf("p1 and p2 are equal.")`

}



`if((p1.x == p2.x) && (p1.y == p2.y))` right way

{

`printf("p1 and p2 are equal.")`

}



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 ." )  
}
```

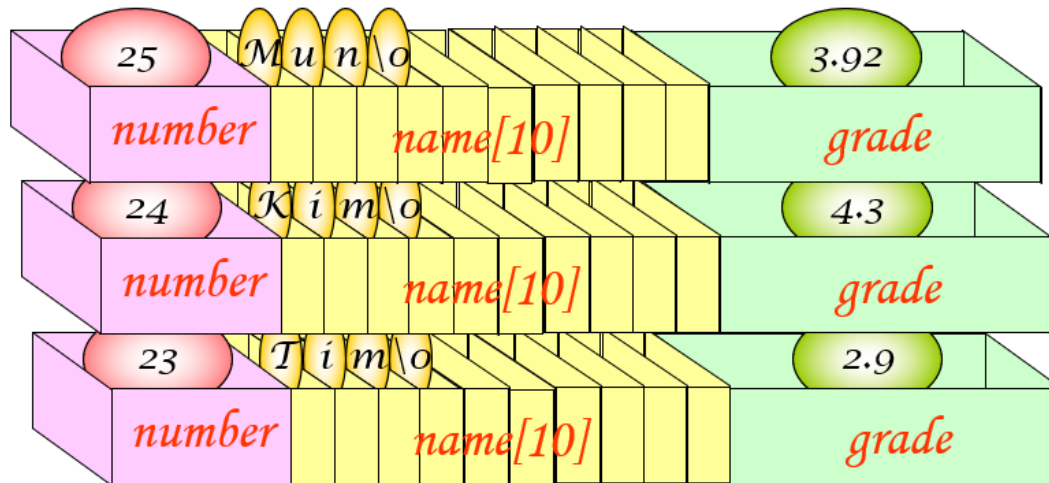
Check points

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);`

This is only possible within the same function .



```
#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

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

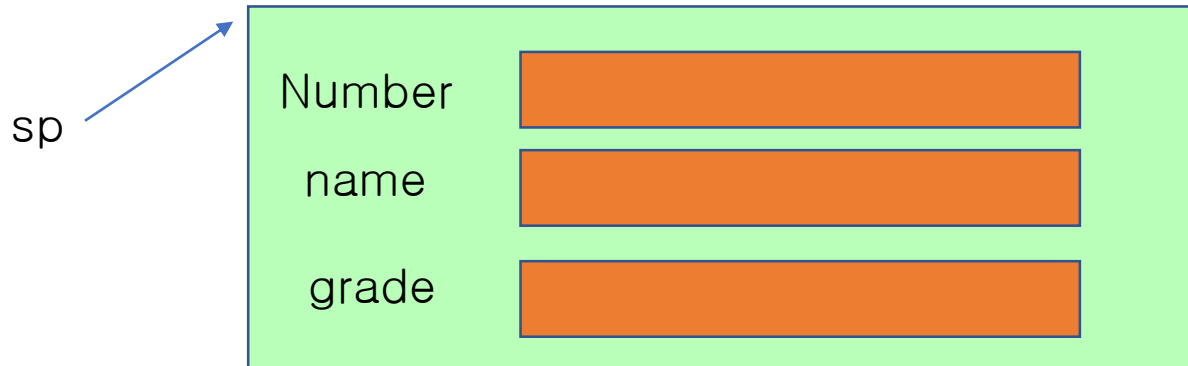
Check points

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;
```

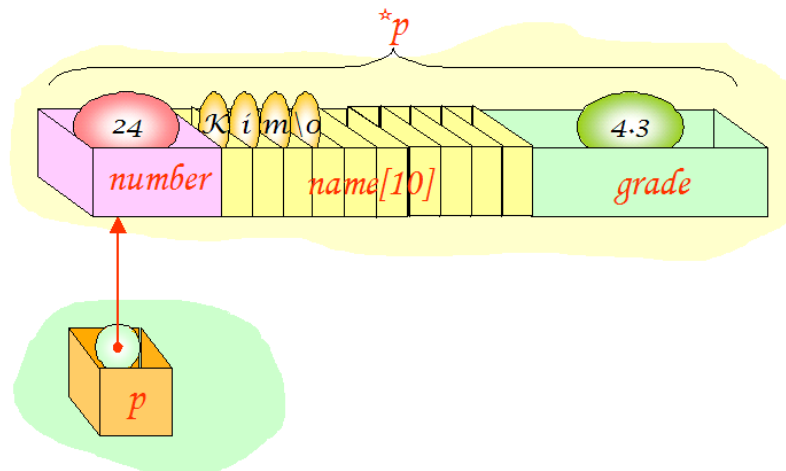
```
struct student s = { 24, "Kim", 4.3 };
```

```
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;
```

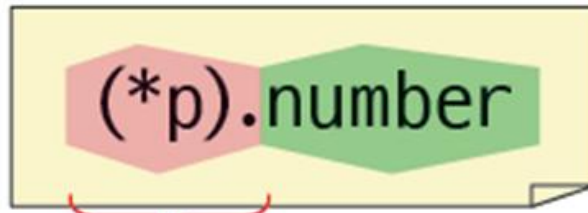
```
struct student s = { 24, "Kim", 4.3 };
```

```
struct student *p;
```

```
p = &s;
```

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

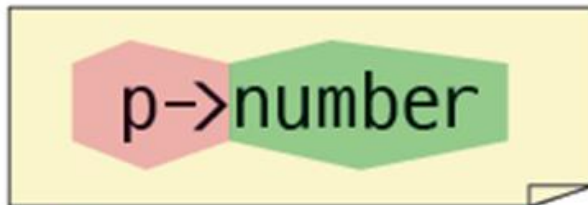
-> Operator



structure variable pointed to by p



Member number of the structure variable pointed to by p



Member number of the structure variable pointed to by p

Example



Practice

```
// 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;
```

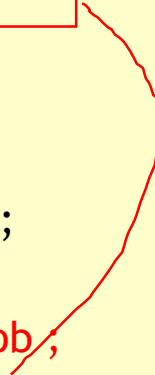
```
}
```

```
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
```

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 *dob;  
};
```



A structure that has pointers as members.

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

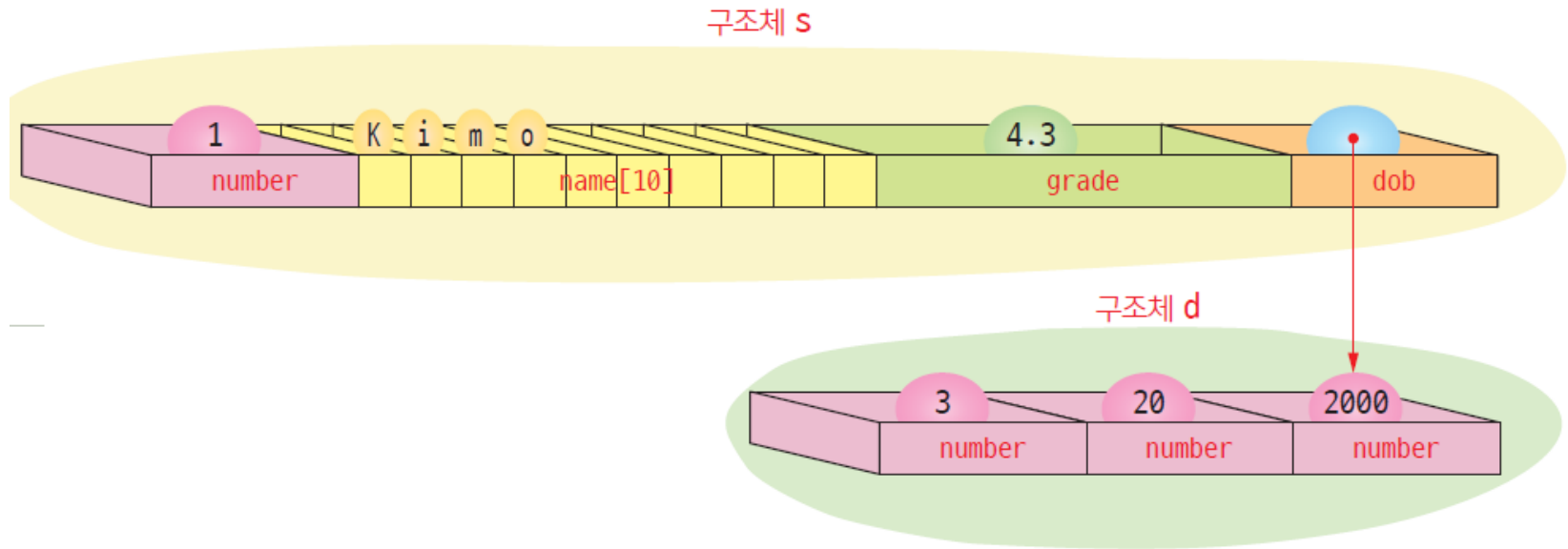
    s.dob = &d;

    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.dob ->year, s.dob ->month, s.dob ->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 .

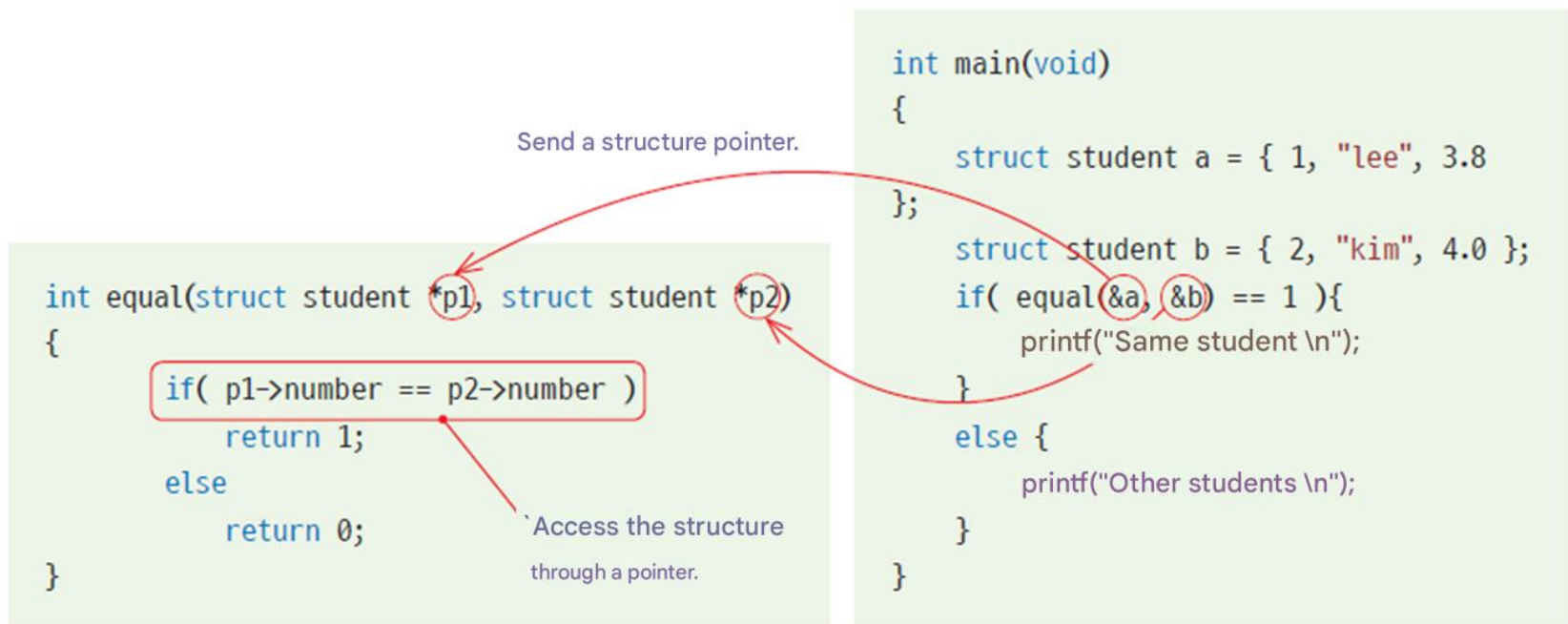
For structures, they are copied.

```
int equal(struct student s1, struct student s2)
{
    if( s1.number == s2.number )
        return 1;
    else
        return 0;
}
```

```
int main(void)
{
    struct student a = { 1, "lee", 3.8 };
    struct student b = { 2, "kim", 4.0 };
    if( equal(a, b) == 1 ){
        printf("Same student \n");
    }
    else {
        printf("Other students \n");
    }
}
```

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 .

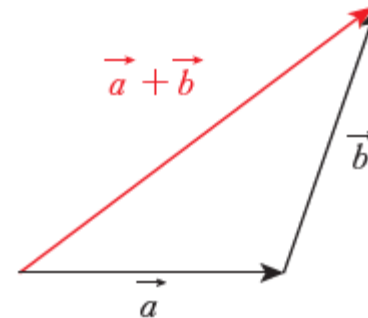
```
struct student create()
{
    struct student s;
    s.number = 3;
    strcpy(s.name, "park");
    s.grade = 4.0;
    return s;
}
```

Structure s is copied into structure a.

```
int main(void)
{
    struct student a;
    a = create();
    return 0;
}
```

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;
}
```

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 modifying the structure ?

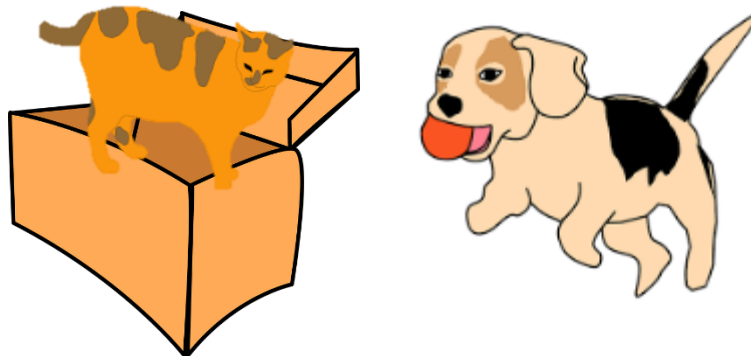


Union

- 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



Example

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

```
union example {  
    int i;  
    char c;  
};
```

Declaration of a union

```
int main( void )  
{  
    union example v;  
  
    v.c = 'A';  
    printf ( "v.c :%c v.i :%i \n" , v.c , v.i );  
  
    vi = 10000;  
    printf ( "v.c :%c v.i :%i \n" , v.c , v.i );  
}
```

Declaring a union variable .

Reference to char type .

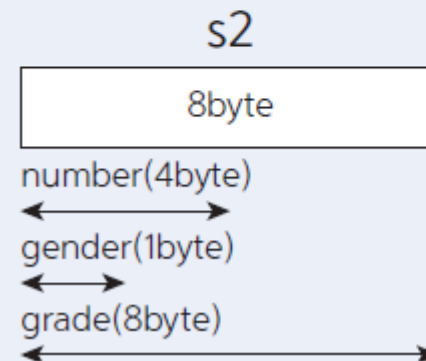
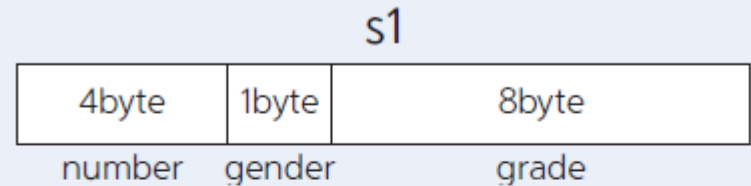
Reference to type int .

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

Difference between structure and union

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

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



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);
}
```

Student number : 20190001
Name : Hong Gil-dong
Resident registration number : 860101-1056076
Name : Kim Cheol-su

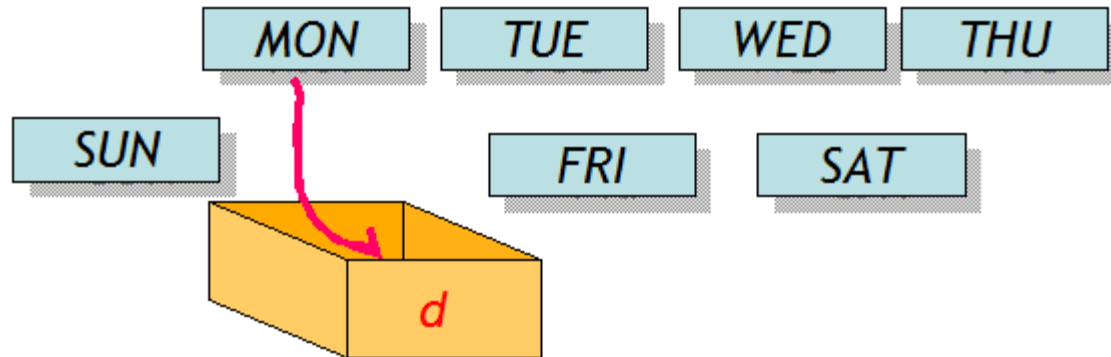
Check points

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



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}



Declaration of enumeration

Syntax

Enum definition

yes

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

When defining an enumeration

Name (tag) of the enumeration

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

Keywords used

```
enum days today;
```

Declaring enumeration variables

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

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;

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 .
 - It is also necessary to block meaningless values such as `9` from being assigned to `today` in advance.

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,
```

...

If you do not specify a value,
it is assigned from 0.

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 >

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

char * days_name [] = {
    "sunday", "monday", "tuesday", "wednesday", "thursday", "friday", "saturday" };

int main( void )
{
    enum days d;
    d = WED;
    printf ( "The % dth day of the week is %s \n" , d, days_name [d]);
    return 0;
}
```

The third day of the week is wednesday

Comparison of enumerations with other methods

Use of integers	Symbolic Constant	Enumeration
<pre>switch (code) { case 1: printf ("LCD TV\n"); break ; case 2: printf ("OLED TV\n"); break ; }</pre>	<pre># define LCD 1 #define OLED 2 switch (code) { case LCD: printf ("LCD TV\n"); break ; case OLED: printf ("OLED TV\n"); break ; }</pre>	<pre>enum tvtype { LCD, OLED }; enum tvtype code; switch (code) { case LCD: printf ("LCD TV\n"); break ; case PDP: printf ("OLED TV\n"); break ; }</pre>
Computers are easy to understand, but people have difficulty remembering .	When writing symbolic constants .	The compiler checks to ensure that no duplication occurs .

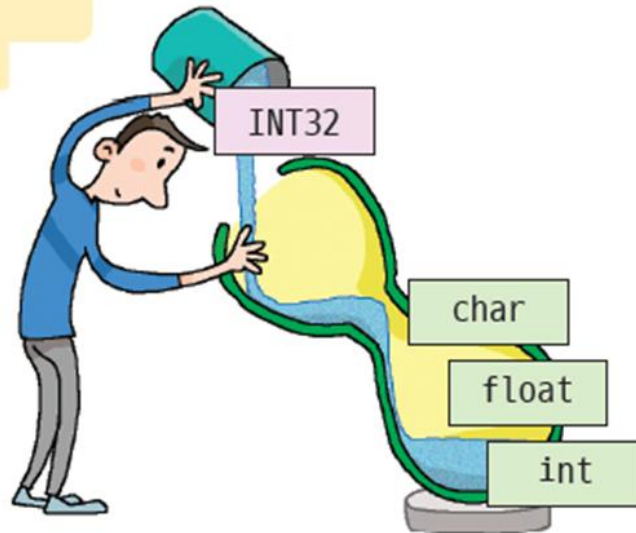
Check points

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 is adding a new data type to the basic data types.



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



typedef

This defines the existing
data type unsigned char as a new data type
BYTE.



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;
```

Comparison of typedef and #define

- Increases portability.
 - Code can be made independent of computer hardware
 - (Example) The int type is 2 bytes or 4 bytes . If you use INT32 or INT16 using typedef instead of the int type, you can clearly specify whether it is 2 bytes or 4 bytes .
- You can achieve a similar effect to typedef by using #define . That is, you can define INT32 as follows :
 - #define UINT32 unsigned int
 - typedef float VECTOR[2];// Not possible with #define .
- It also serves as documentation .
 - Using typedef has the same effect as adding a comment.

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

