# 計算機程式設計

## Computer Programming

## Structures

**Instructor: 林英嘉**

**2024/12/09**

W14 Slido: #3222730

GitHub repo

# Outline

- What are structures?
- struct
- typedef
- Passing a Structure to a Function

# C keywords

- This is a list of reserved keywords in C (保留字). Since they are used by the language, these keywords are not available for re-definition.

| | | | |
|---|---|---|---|
| alignas (C23) | extern | sizeof | _Alignas (C11)(deprecated in C23) |
| alignof (C23) | false (C23) | static | _Alignof (C11)(deprecated in C23) |
| auto | float | static_assert (C23) | _Atomic (C11) |
| bool (C23) | for | struct | _BitInt (C23) |
| break | goto | switch | _Bool (C99)(deprecated in C23) |
| case | if | thread_local (C23) | _Complex (C99) |
| char | inline (C99) | true (C23) | _Decimal128 (C23) |
| const | int | typedef | _Decimal32 (C23) |
| constexpr (C23) | long | typeof (C23) | _Decimal64 (C23) |
| continue | nullptr (C23) | typeof_unqual (C23) | _Generic (C11) |
| default | register | union | _Imaginary (C99) |
| do | restrict (C99) | unsigned | _Noreturn (C11)(deprecated in C23) |
| double | return | void | _Static_assert (C11)(deprecated in C23) |
| else | short | volatile | _Thread_local (C11)(deprecated in C23) |
| enum | signed | while | |

https://en.cppreference.com/w/c/keyword

# [Definition] What are structures?

- A structure is a logical choice for storing a collection of related data items.
- The elements of a structure (its members) aren't required to have the same type.
- C structures can be used with the **struct** keyword:

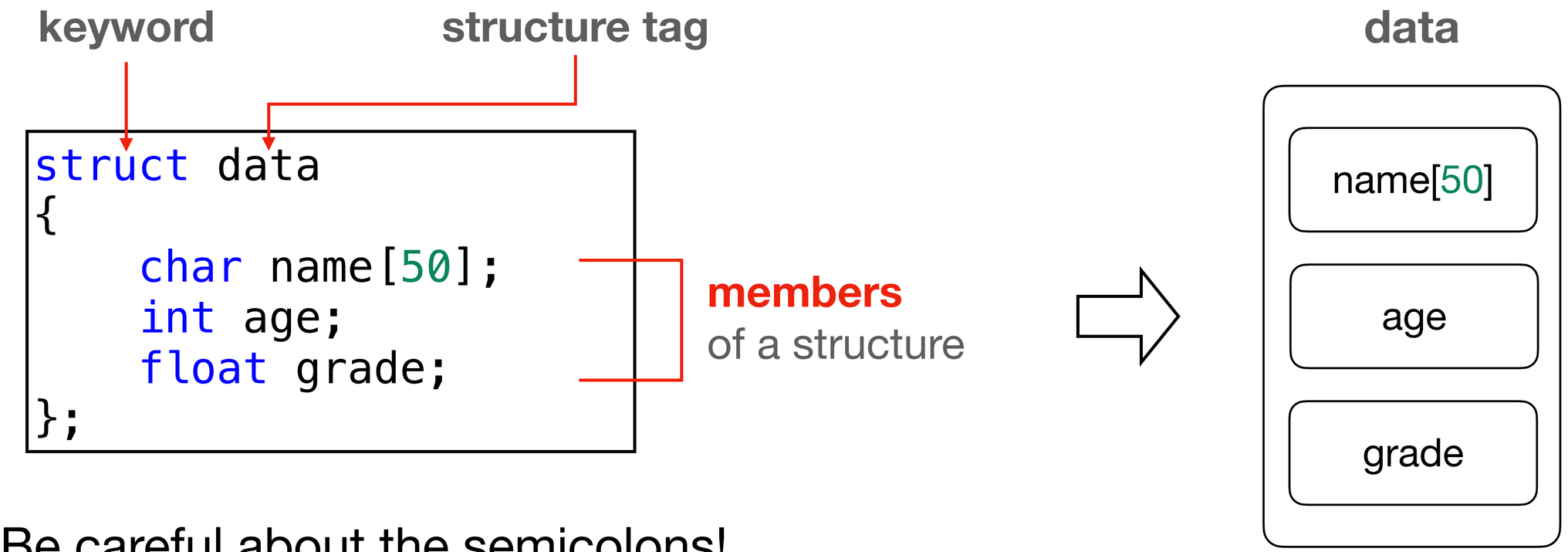
```
struct structure_tag
{
    type1 name1;
    type2 name2;
    ...
    typeN nameN;
};
```

**members** of a structure

**structure tag (type_name): a name for the structure type**

# [Usage] What are structures?

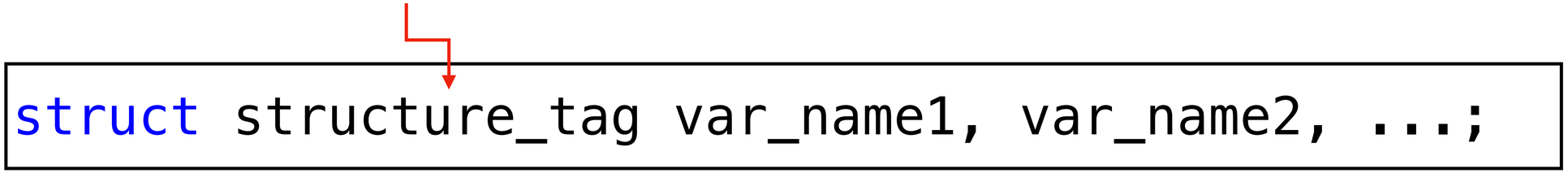
- Declare a structure:




- Be careful about the semicolons!

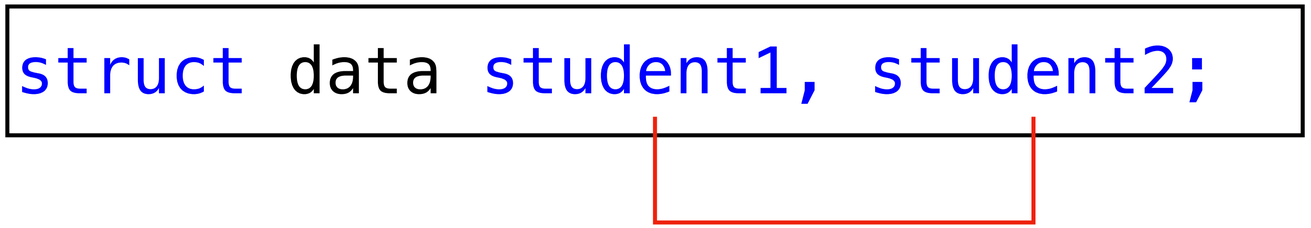
# [Declaration] Create a Variable with a Structure

We need to first declare a structure!



```
struct structure_tag var_name1, var_name2, ...;
```

Take `data` we declare in the previous page as an example:



```
struct data student1, student2;
```

The object_names, `student1` and `student2`, have the same structure, with the **data type** called **struct data**.

# [Illustration] Structure Variable Declaration

**Steps:**

1. Declare a structure

```
struct data
{
    char name[50];
    int age;
    float grade;
};
```

2. Declare a structure variable

```
struct data student1, student2;
```

# [Usage] Dot Operator

- The C dot (.) operator is used for direct member selection via the name of variables of type struct and union (not included in this course).
- To access a member within a structure, we write the name of the structure first, then a period, then the name of the member.

```
object_name.member;
```

- Example from the previous page:

```
printf("Name: %s\n", student1.name);
```

# An Example of Structure Declaration

C-course-materials/08-Structures/basic_declaration1.c

```c
#include <stdio.h>
int main(void) {
    struct data
    {
        char name[50];
        float grade;
    };
    struct data student1;
    printf("Enter name: ");
    scanf("%s", student1.name);
    printf("Enter grade: ");
    scanf("%f", &student1.grade);
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
}
```

- Input

```
John
98.5
```

- Output

```
Name: John
Grade: 98.500000
```

# [Important Notes] Dot Operator

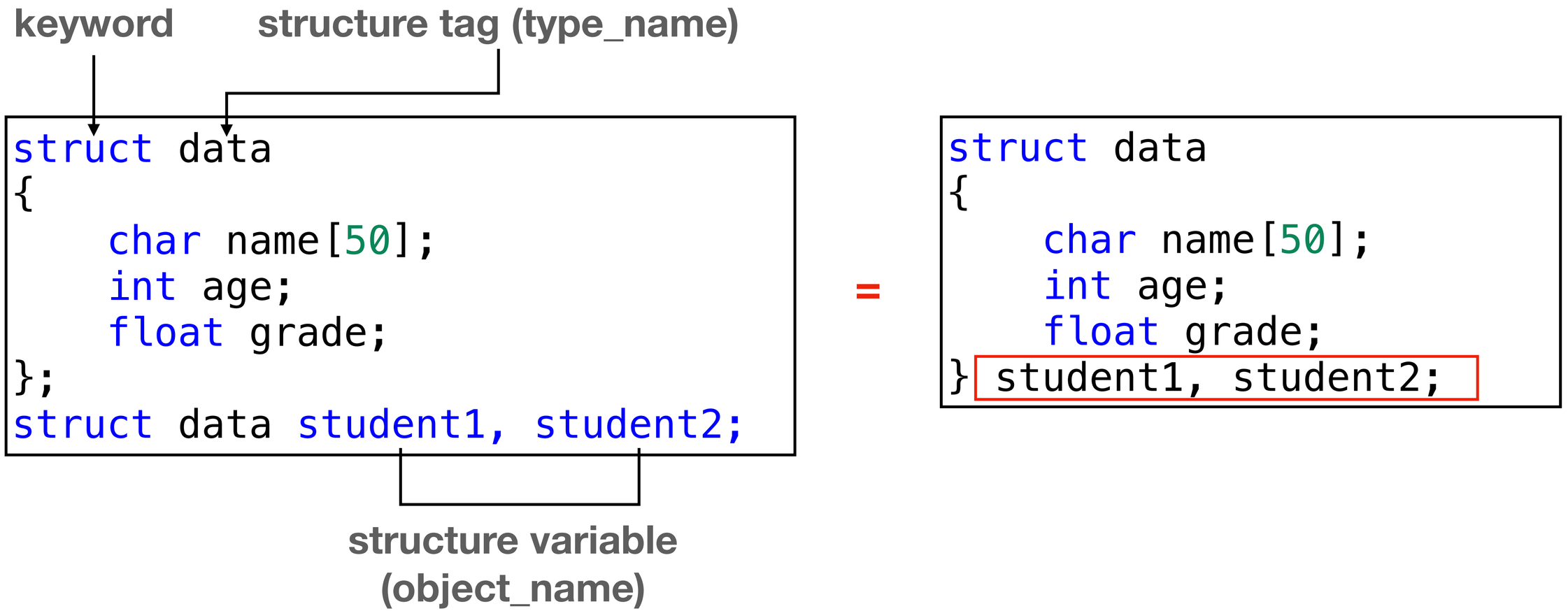
- The period takes precedence over nearly all other operators.
- Example:

```
scanf("%d", &student1.age);
```

- The . operator takes precedence over the & operator, so & computes the address of student1.age.

# [Usage] Declare Structure Variable right after a Structure

- Declare a structure:



# Examples of Structure Declaration

C-course-materials/08-Structures/basic_declaration1.c

C-course-materials/08-Structures/basic_declaration2.c

```c
#include <stdio.h>
int main(void) {
    struct data {
        char name[50];
        float grade;
    };
    struct data student1;
    printf("Enter name: ");
    scanf("%s", student1.name);
    printf("Enter grade: ");
    scanf("%f", &student1.grade);
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
}
```

```c
#include <stdio.h>
int main(void) {
    struct data {
        char name[50];
        float grade;
    } student1;

    printf("Enter name: ");
    scanf("%s", student1.name);
    printf("Enter grade: ");
    scanf("%f", &student1.grade);
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
}
```

You don’t need to declare the structure variable again in the right code.

# A structure tag can be optional

C-course-materials/08-Structures/basic_declaration2.c

- When a structure variable is specified, the structure tag (data) becomes optional.
- struct requires either a type_name or at least one name in object_names, but not necessarily both.

```c
#include <stdio.h>
int main(void) {
    struct data {
        char name[50];
        float grade;
    } student1;

    printf("Enter name: ");
    scanf("%s", student1.name);
    printf("Enter grade: ");
    scanf("%f", &student1.grade);
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
}
```

https://cplusplus.com/doc/tutorial/structures/

# Summary

- Given this example, we can summarize the different compositions in this code:

```c
#include <stdio.h>
int main(void) {
    struct data {
        char name[50];
        float grade;
    } student1;
}
```

struct data: structure type (data type)

data: structure tag

name: a member of struct data

grade: a member of struct data

**student1: structure variable**

# Initializing a Structure

C-course-materials/08-Structures/struct_init1.c

C-course-materials/08-Structures/struct_init2.c

- We can use an initializer with braces {} to assign the values for a structure member.

```c
#include <stdio.h>
int main(void) {
    struct data
    {
        char name[50];
        float grade;
    };
    struct data student1 = {"John", 98.5};
    struct data student2 = {"Mary", 100.0};
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
    printf("Name: %s\n", student2.name);
    printf("Grade: %f\n", student2.grade);
}
```

```c
#include <stdio.h>
int main(void) {
    struct data
    {
        char name[50];
        float grade;
    } student1 = {"John", 98.5},
      student2 = {"Mary", 100.0};
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
    printf("Name: %s\n", student2.name);
    printf("Grade: %f\n", student2.grade);
}
```

# [Important Notes] Initializing a Structure

- Structure initializers follow rules similar to those for array initializers.
- Expressions used in a structure initializer must be constant.
  - (This restriction is relaxed in C99.)
- An initializer can have fewer members than the structure it's initializing.
- Any "leftover" members are given 0 as their initial value.

# Designated Initializers

C-course-materials/08-Structures/struct_designated.c

- Values in a designated initializer don’t have to be placed in the same order that the members are listed in the structure.

```c
#include <stdio.h>
int main(void) {
    struct data
    {
        char name[50];
        float grade;
    }
    // designated initializers
    student1 = {.grade = 98.5, .name = "John"},
    student2 = {.name = "Mary", .grade = 100.0};
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
    printf("Name: %s\n", student2.name);
    printf("Grade: %f\n", student2.grade);
}
```

# Copy Members: Assignment

C-course-materials/08-Structures/struct_assignment.c

```c
#include <stdio.h>
int main(void) {
    struct data
    {
        char name[50];
        float grade;
    } student1 = {"John", 98.5};
    struct data student2;
    student2 = student1;
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
    printf("Name: %s\n", student2.name);
    printf("Grade: %f\n", student2.grade);
}
```

- The effect of **assignment** is to copy the members from A structure to B structure.
- In this case, the values of the members in `student2` will be as same as the ones in `student1`.

# [Important Notes] The = operator

- The = operator can be used only with structure variables declared from the same structure type.
  - Structures declared using the same “structure tag”.
- Other than assignment, C provides no operations on entire structures.
- Operators == and != can’t be used with structures.

# Initialization and Assignment at the Same Time

C-course-materials/08-Structures/struct_init_assignment.c

```c
// Initialization and assignment at the same time
#include <stdio.h>
int main(void) {
    struct data
    {
        char name[50];
        float grade;
    } student1 = {"John", 98.5},
    student2 = student1;
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
    printf("Name: %s\n", student2.name);
    printf("Grade: %f\n", student2.grade);
}
```

# [Usage] Nested Structures

```
struct Structure_1
{
    /* Members of Structure_1*/
};
struct Structure_2
{
    /* Members of Structure_2*/
    struct Structure_1 var_name_s1;
};
```

Declare a structure variable
in another structure

Be careful about the sequence!
In this case, we must declare struct Structure1 first.

# Nested Structures

C-course-materials/08-Structures/nested_struct.c

```c
#include <stdio.h>
int main(void) {
    struct date
    {
        int month;
        int day;
    };
    struct Student1
    {
        char name[50];
        float grade;
        struct date birthday;
    };
    struct Student1 student1 = {"John", 98.5, {12, 25}};
    printf("Name: %s\n", student1.name);
    printf("Grade: %.1f\n", student1.grade);
    printf("Birthday: %d/%d\n", student1.birthday.month, student1.birthday.day);
}
```

# Nested Structures (with an initializer)

C-course-materials/08-Structures/nested_struct.c

- We can also use an initializer with braces {} to assign the values for a structure member.

```c
#include <stdio.h>
int main(void) {
    struct date
    {
        int month;
        int day;
    };
    struct Student1
    {
        char name[50];
        float grade;
        struct date birthday;
    } student1 = {"John", 98.5, {12, 25}};
    printf("Name: %s\n", student1.name);
    printf("Grade: %.1f\n", student1.grade);
    printf("Birthday: %d/%d\n", student1.birthday.month, student1.birthday.day);
}
```

# [Usage] Arrays of Structures

- Before creating an array of structures, we need to declare a structure name first.

```
struct structure_name var_name[length];
```

- Example:

```
struct Student {
    char name[50];
    float grade;
};
struct Student students[10];
```

length=10

| name | grade |
| --- | --- |
| students[0].name | students[0].grade |
| students[1].name | students[1].grade |
| ... | |
| students[9].name | students[9].grade |

# Check the size for an Array of Structure

C-course-materials/08-Structures/arr_struct_size.c

```c
#include <stdio.h>
int main() {
    struct Student {
        char name[50];
        float grade;
    };
    struct Student students[10];
    printf("%d\n", sizeof(students));
    printf("%d\n", sizeof(students[0]));
}
```

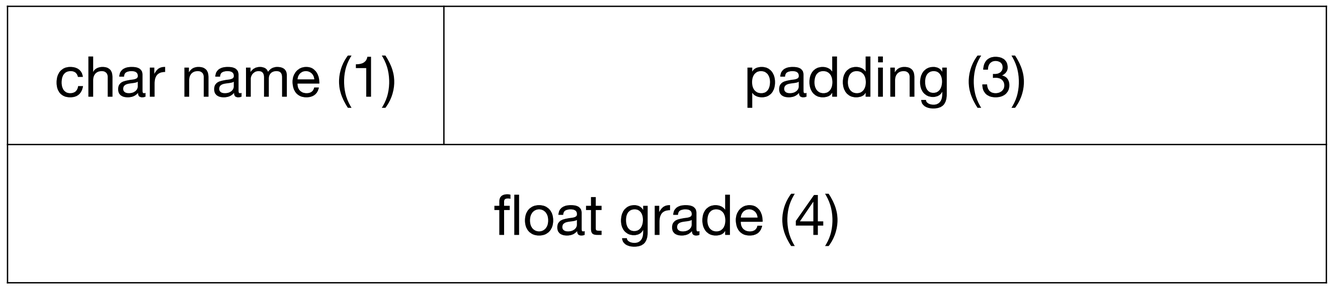
- 50 -> 52 for memory padding
- Each float variable occupies 4 bytes.

# [Illustration] Memory Allocation for a Structure

```c
int main(void) {
    struct Student {
        char name;
        float grade;
    };
    struct Student student1;
}
```

- struct will automatically performs memory alignment, which may add padding between members to ensure that each member is aligned for faster memory access.
- The size of the structure is a **multiple** of the largest alignment requirement among its members.

**student1**
(total size: 8)



# An Example for Arrays of Structures

C-course-materials/08-Structures/arr_struct_example.c

```c
#include <stdio.h>
struct Student {
    char name[50];
    float grade;
};
int main() {
    struct Student students[10] = {
        {"John", 98.5},
        {"Alice", 87.0},
        {"Bob", 92.3},
        {"Mary", 76.4}
    };
    int studentCount = sizeof(students) / sizeof(students[0]);
    printf("學生名單：\n");
    for (int i = 0; i < studentCount; i++) {
        // break when no students
        if (students[i].name[0] == '\0') {
            break;
        }
        printf("姓名: %s, 成績: %.1f\n", students[i].name, students[i].grade);
    }
}
```

# [Usage] typedef

- typedef: **type def**inition
- Define a custom data type for an existing data type

```
typedef data_type YOUR_TYPE_NAME;
```

- This method is just like to **give a nickname** to an existing data type.

# typedef Example for a Regular Data Type

```c
#include <stdio.h>
typedef char new_type[50];

int main(void) {
    char string_a[50] = "Hello World!";
    new_type string_b = "Hello World!";
    printf("String A: %s\n", string_a);
    printf("String B: %s\n", string_b);
    printf("Size of string_b: %d\n", sizeof(string_b));
}
```

In this case, `new_type` can be reused for a char array with a length of 50, but we cannot use `new_type` for a length different from 50.

- Output

```
String A: Hello World!
String B: Hello World!
Size of string_b: 50
```

# typedef Example for a Structure

C-course-materials/08-Structures/struct_init1.c
C-course-materials/08-Structures/typedef_struct.c

```c
#include <stdio.h>
int main(void) {
    struct Data
    {
        char name[50];
        float grade;
    };
    struct Data student1 = {"John", 98.5};
    struct Data student2 = {"Mary", 100.0};
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
    printf("Name: %s\n", student2.name);
    printf("Grade: %f\n", student2.grade);
}
```

```c
#include <stdio.h>
struct Data
{
    char name[50];
    float grade;
};
typedef struct Data Student;

int main(void) {
    Student student1 = {"John", 98.5};
    Student student2 = {"Mary", 100.0};
    printf("Name: %s\n", student1.name);
    printf("Grade: %f\n", student1.grade);
    printf("Name: %s\n", student2.name);
    printf("Grade: %f\n", student2.grade);
}
```

With typedef, we don't need a struct keyword when declaring a structure variable. We can just use the nickname you set as the data type.

# Pointers to Structures
## (Structure Pointers)

# [Recap, Declaration] Pointers

- Pointer declaration: Data_type * Variable_name

  Data_type * is a data type for pointers

- When a pointer variable is declared, its name must be preceded by an **asterisk**:

```
int *iptr; // A pointer points to an int variable (整數指標)

float *fptr; // A pointer points to a float variable (浮點數指標)

double *dfptr; // A pointer points to a double variable (倍準浮點數指標)
```

# [Declaration] A Pointer to a Structure

```
struct struct_data_type *pointer_name;
```

- Example:

```
struct data {
    char name[50];
    float grade;
} student1;

struct data *ptr;
ptr = &student1;
```

```
struct data *ptr = &student1;
```

# [Usage] Get Content from a Structure Pointer

- We can use -> to access the member in a structure with the structure pointer.

```
pointer_name->member_name;
```

- This operation is similar to dereference with an asterisk:

```
(*pointer_name).member_name;
```

- Please note that we must use parentheses here because the . operator takes precedence.

# Get Content from a Structure Pointer

C-course-materials/08-Structures/struct_pointer.c

- We can use -> to access the member in a structure with the structure pointer.

```c
#include <stdio.h>
int main() {
    struct data {
        char name[50];
        float grade;
    } student1;

    struct data *ptr;
    ptr = &student1; // struct data *ptr = &student1;
    printf("Enter name: ");
    scanf("%s", student1.name);
    printf("Enter grade: ");
    scanf("%f", &student1.grade);
    printf("Name: %s\n", ptr->name);
    printf("Grade: %f\n", ptr->grade);
    printf("Name: %s\n", (*ptr).name);
    printf("Grade: %f\n", (*ptr).grade);
}
```

# Passing a Structure to a Function

# [Usage] Passing a Structure to a Function

1. **(By Value) Pass the whole structure to a function**
   - Like integers, float, or other regular variables
2. **(By Reference) Pass the address of a structure to a function**
   - Like arrays or pointers

# [Usage] Passing a Structure to a Function

1. **Pass the whole structure to a function**

```
return_type func_name(struct type_name in_obj_name) {
   // Descriptions
}

int main(void){
    struct type_name out_obj_name;
    func_name(out_obj_name);
}
```

# Passing a Structure to a Function

C-course-materials/08-Structures/struct_to_func_1.c

## 1. Pass the whole structure to a function

```c
#include <stdio.h>
struct data {
    char name[50];
    float grade;
};
// Function to display a student's info
void displayStudent(struct data s) {
    printf("Student Details (By Value):\n");
    printf("Student name: %s\n", s.name);
    printf("Student grade: %.1f\n", s.grade);
}
int main(void){
    struct data student1 = {"John", 98.5};
    displayStudent(student1);
}
```

# [Wrong!] Passing a Structure to a Function

C-course-materials/08-Structures/struct_to_func_1.c

## 1. Pass the whole structure to a function

```c
#include <stdio.h>
// Function to display a student's info
void displayStudent(struct data s) {
    printf(“Student’s info (By Value):\n");
    printf("Student name: %s\n", s.name);
    printf("Student grade: %.1f\n", s.grade);
}
struct data {
    char name[50];
    float grade;
};
int main(void){
    struct data student1 = {"John", 98.5};
    displayStudent(student1);
}
```

Please note the declaration sequence.

In this case, it is incorrect the declare `struct data` after `displayStudent`.

# [Usage] Passing a Structure to a Function

**2. Pass the address of a structure to a function**

```
return_type func_name(struct type_name *ptr_name) {
   // Descriptions
}

int main(void){
    struct type_name out_obj_name;
    func_name(&out_obj_name);
}
```

# Passing a Structure to a Function and modify the string

C-course-materials/08-Structures/struct_to_func_2.c

## 2. Pass the address of a structure to a function

```c
#include <stdio.h>
struct data {
    char name[50];
    float grade;
};
```

```c
#include <string.h>
void modifyStudent(struct data *s) {
    // incorrect: s->name = "Daniel";
    strcpy(s->name, "Daniel"); // Assign a new name
    s->grade = 90.5; // Assign a new grade
    printf("Student Details Modified (By Reference):\n");
    printf("Name: %s\n", s->name);
    printf("New Grade: %.1f\n", s->grade);
}
int main(void){
    struct data student1 = {"John", 98.5};
    modifyStudent(&student1);
}
```

# Other content (not included in this course)

- enum
- Unions