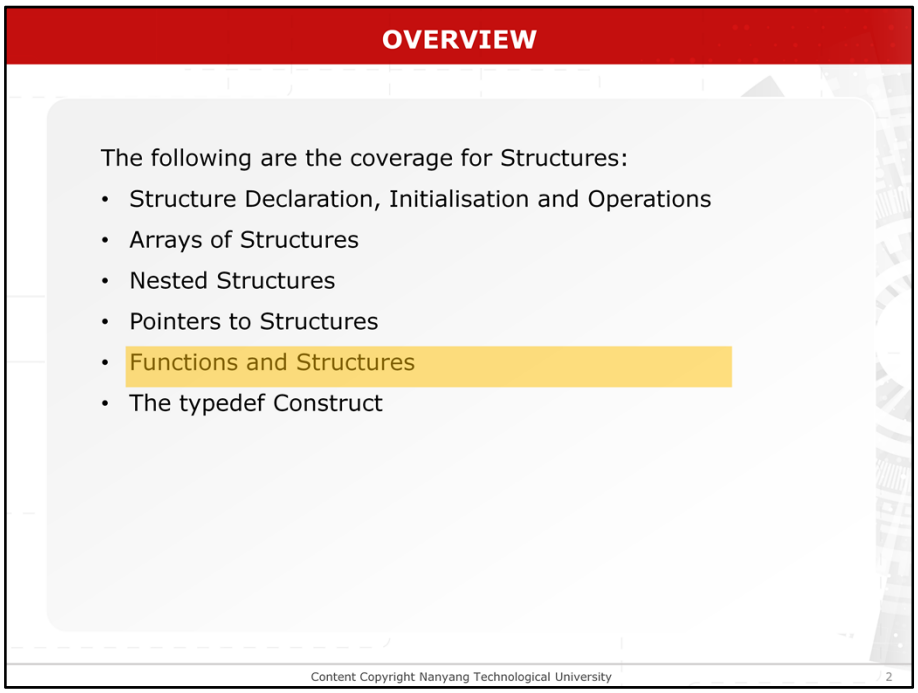
NANYANG TECHNOLOGICAL UNIVERSITY

CE1007/ CZ1007 DATA STRUCTURES

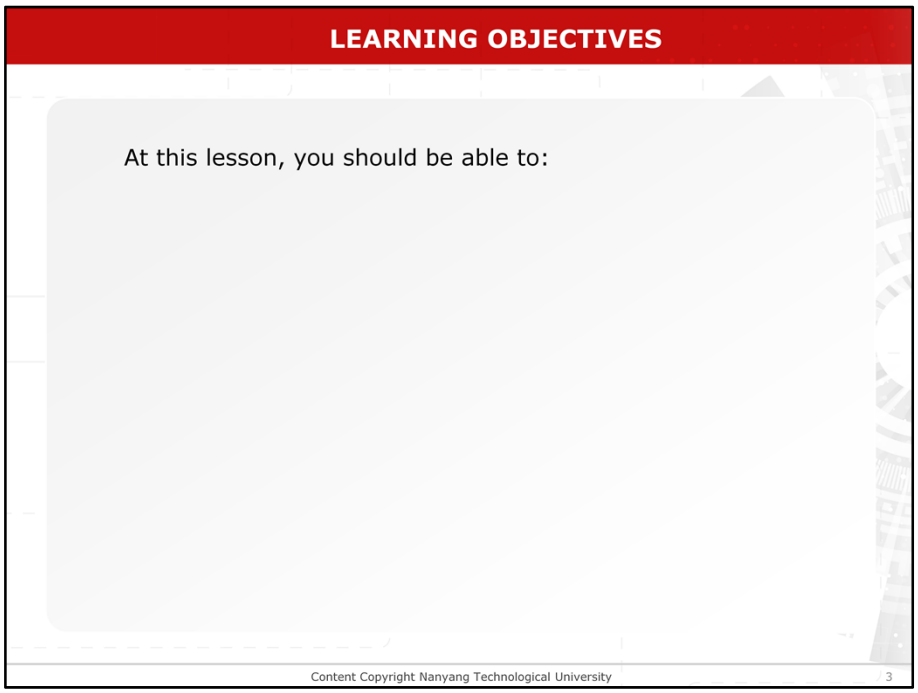
Lesson 9.5 Functions and Structures

Assoc Prof Hui Siu Cheung

College of Engineering
School of Computer Science and Engineering



The following are the coverage for Structures: this video focusses on Pointers to Structures.



LEARNING OBJECTIVES: At this lesson, you should be able to:

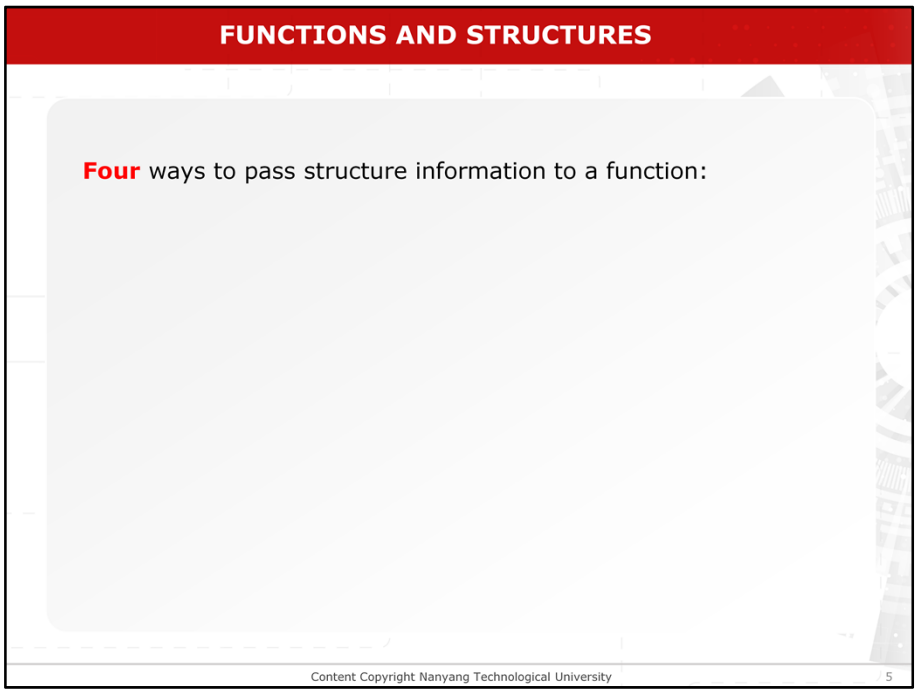
LEARNING OBJECTIVES

At this lesson, you should be able to:

- Explain the different ways to pass structure information to a function.

 4

Explain the different ways to pass structure information to a function.



Functions and Structures

It is often necessary to pass structure information to a function. In C, there are four ways to pass structure information to a function:

FUNCTIONS AND STRUCTURES

Four ways to pass structure information to a function:

1. Passing **structure members** as arguments;

Passing structure members as arguments

FUNCTIONS AND STRUCTURES

Four ways to pass structure information to a function:

1. Passing **structure members** as arguments;
2. Passing **structures** as arguments;

 7

Passing structures as arguments;

FUNCTIONS AND STRUCTURES

Four ways to pass structure information to a function:

1. Passing **structure members** as arguments;
2. Passing **structures** as arguments;
3. Passing **pointers to structures** as arguments;

 8

Passing pointers to structures as arguments; and

FUNCTIONS AND STRUCTURES

Four ways to pass structure information to a function:

1. Passing **structure members** as arguments;
2. Passing **structures** as arguments;
3. Passing **pointers to structures** as arguments;
4. Passing by **returning structures**.

 9

Passing by returning structures.

FUNCTIONS AND STRUCTURES

Four ways to pass structure information to a function:

1. Passing **structure members** as arguments;
2. Passing **structures** as arguments;
3. Passing **pointers to structures** as arguments;
4. Passing by **returning structures**.

Basically, parameter passing between functions using structure is **similar** to other basic data types such as int, float, etc.

 10

Basically, parameter passing between functions using structure is similar to other basic data types such as **int**, **float**, etc.

(1) PASSING STRUCTURE MEMBERS AS ARGUMENTS

```
#include <stdio.h>
float sum(float, float);
struct account {
   char  bank[20];
   float  current;
   float  saving;
};
int main( )
{
   struct account john={"OCBC Bank",1000.43, 4000.87};
   printf("The account has a total of %.2f.\n",
        sum(john.current, john.saving));          // pass by value
   return 0;
}
float sum(float x, float y)
{
   return (x+y);
}
```

 11

Passing Structure Members as Arguments

In the program, a structure template **account** is defined with three members: **bank**, **current** and **saving**.

(1) PASSING STRUCTURE MEMBERS AS ARGUMENTS

```
#include <stdio.h>
float sum(float, float);
struct account {
  char  bank[20];
  float  current;
  float  saving;
};
int main( )
{
  struct account john={"OCBC Bank",1000.43, 4000.87};
  printf("The account has a total of %.2f.\n",
       sum(john.current, john.saving));          // pass by value
  return 0;
}
float sum(float x, float y)
{
  return (x+y);
}
```

 12

In the **main()** function, an **account** structure variable **john** is declared with initial values.

(1) PASSING STRUCTURE MEMBERS AS ARGUMENTS

```
#include <stdio.h>
float sum(float, float);
struct account {
  char  bank[20];
  float  current;
  float  saving;
};
int main( )
{
  struct account john={"OCBC Bank",1000.43, 4000.87};
  printf("The account has a total of %.2f.\n",
       sum(john.current, john.saving));          // pass by value
  return 0;
}
float sum(float x, float y)
{
  return (x+y);
}
```

 13

The function **sum()** is used to compute the total amount from the saving and current accounts. There are different ways to implement the function **sum()**.

(1) PASSING STRUCTURE MEMBERS AS ARGUMENTS

```
#include <stdio.h>
float sum(float, float);
struct account {
  char  bank[20];
  float  current;
  float  saving;
};
int main( )
{
  struct account john={"OCBC Bank",1000.43, 4000.87};
  printf("The account has a total of %.2f.\n",
       sum(john.current, john.saving));          // pass by value
  return 0;
}
float sum(float x, float y)
{
  return (x+y);
}
```

- **struct members** are used as arguments

 14

The first approach is to pass individual members of a structure as arguments to a function.

(1) PASSING STRUCTURE MEMBERS AS ARGUMENTS

```
#include <stdio.h>
float sum(float, float);
struct account {
  char  bank[20];
  float  current;
  float  saving;
};
int main( )
{
  struct account john={"OCBC Bank",1000.43, 4000.87};
  printf("The account has a total of %.2f.\n",
       sum(john.current, john.saving));          // pass by value
  return 0;
}
float sum(float x, float y)
{
  return (x+y);
}
```

- **struct members** are used as arguments

 15

in the program, the function **sum()** expects two arguments, **x** and **y**, of type **float**. The structure variable **john** is declared with **struct account** and the values of the members **current** and **saving** are passed to the function **sum()**.

(1) PASSING STRUCTURE MEMBERS AS ARGUMENTS

```
#include <stdio.h>
float sum(float, float);
struct account {
  char  bank[20];
  float  current;
  float  saving;
};
int main( )
{
  struct account john={"OCBC Bank",1000.43, 4000.87};
  printf("The account has a total of %.2f.\n",
       sum(john.current, john.saving));          // pass by value
  return 0;
}
float sum(float x, float y)
{
  return (x+y);
}
```

- **struct members** are used as arguments

 16

The structure members **john.current** and **john.saving** are of type **float**.

(1) PASSING STRUCTURE MEMBERS AS ARGUMENTS

```
#include <stdio.h>
float sum(float, float);
struct account {
  char  bank[20];
  float  current;
  float  saving;
};
int main( )
{
  struct account john={"OCBC Bank",1000.43, 4000.87};
  printf("The account has a total of %.2f.\n",
       sum(john.current, john.saving));          // pass by value
  return 0;
}
float sum(float x, float y)
{
  return (x+y);
}
```

Output
The account has a total of 5001.30.

- **struct members** are used as arguments
- Call by value

 17

As long as a structure member is a variable of a data type with a single value, we can pass the structure member as a function argument. The structure members **john.current** and **john.saving** are passed by value. However, it is also possible to pass the structure members using call by reference.

(2) PASSING STRUCTURE AS ARGUMENT

```
#include <stdio.h>
struct account{
    char  bank[20];
    float  current;
    float  saving;
};
float sum(struct account);          /* argument - structure */
int main( )
{
    struct account john = {"OCBC Bank", 1000.43, 4000.87};
    printf("The account has a total of %.2f.\n", sum(john));// pass by value
    return 0;
}
float sum( struct account money)
{
    return(money.current + money.saving);
    /* not money->current */
}
```

- Call by value
- struct account money is used as parameter

Passing Structures as Arguments

The second approach is to pass a structure to a function as an argument to a function. When a structure is passed as an argument to a function, it is passed using call by value. The members of this structure in the function **sum()** are initialized with local copies.

(2) PASSING STRUCTURE AS ARGUMENT

```
#include <stdio.h>
struct account{
    char  bank[20];
    float  current;
    float  saving;
};
float sum(struct account);          /* argument - structure */
int main( )
{
    struct account john = {"OCBC Bank", 1000.43, 4000.87};
    printf("The account has a total of %.2f.\n", sum(john));// pass by value
    return 0;
}
float sum( struct account money)
{
    return(money.current + money.saving);
    /* not money->current */
}
```

Output
The account has a total of 5001.30.

- **Call by value**
- **struct account money** is used as parameter

 19

Notice that we simply use the member operator as shown.

(2) PASSING STRUCTURE AS ARGUMENT

```
#include <stdio.h>
struct account{
    char  bank[20];
    float  current;
    float  saving;
};
float sum(struct account);          /* argument - structure */
int main( )
{
    struct account john = {"OCBC Bank", 1000.43, 4000.87};
    printf("The account has a total of %.2f.\n", sum(john));// pass by value
    return 0;
}
float sum( struct account money)
{
    return(money.current + money.saving);
    /* not money->current */
}
```

Output
The account has a total of 5001.30.

- **Call by value**
- **struct account money** is used as parameter

 20

The advantage of using this method is that the function cannot modify the members of the original structure variables, which is safer than working with the original variables. However, this method is quite inefficient to pass large structures to functions. In addition, it also takes time and additional storage to make a local copy of the structure.

(3) PASSING STRUCTURE ADDRESS AS ARGUMENT

```
#include <stdio.h>
struct account{
   char   bank[20];
   float  current;
   float  saving;
};
float sum(struct account*); /* argument is a structure */

int main()
{
   struct account john = {"OCBC Bank", 1000.43, 4000.87};
   printf("The account has a total of %.2f.\n", sum(&john));
   return 0;
}
float sum(struct account *money)
{
   return( money->current + money->saving);
}
```


Passing Structure Address

The third approach is to pass the address of the structure as an argument. Using the same structure template **account**, in the program, the **sum()** function uses a pointer to a structure account as its argument. The address of **john** is passed to the function that causes the pointer **money** to point to the structure **john**. The **->** operator is then used in the following statement to obtain the values of **john.current** and **john.saving**. This allows the function to access the structure variable and to modify its content. This is a better approach than passing structures as arguments.

(4) RETURNING A STRUCTURE IN FUNCTION

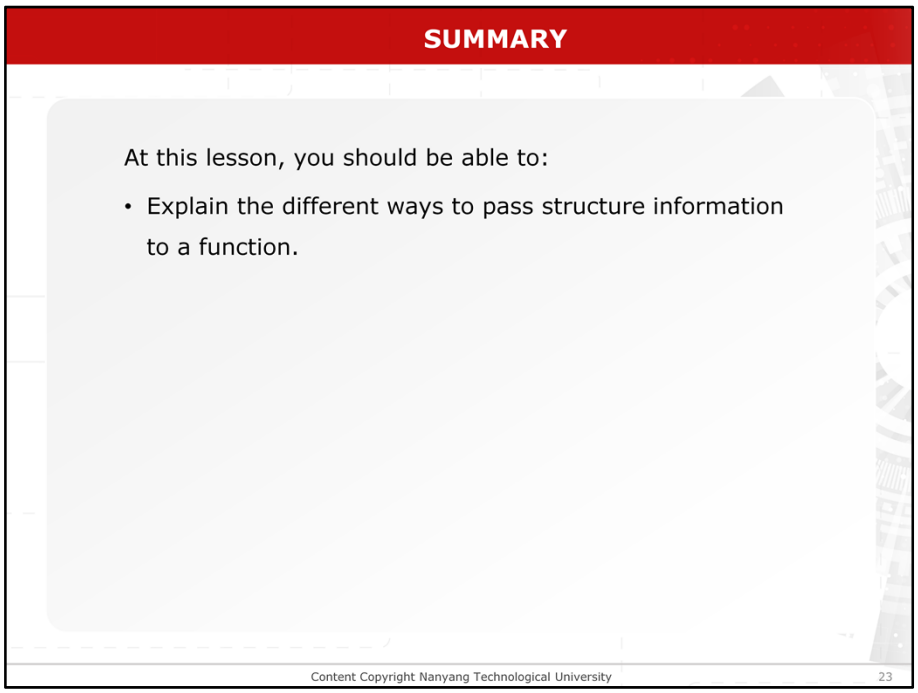
```
#include <stdio.h>
struct nameTag {
   char fname[20], lname[20];
};
struct nameTag getname();

int main()
{
   struct nameTag name;
   name = getname();
   printf("Your name is %s %s\n", name.fname, name.lname);
   return 0;
}
struct nameTag getname()
{
   struct nameTag newname;
   printf("Enter the first name: ");
   gets(newname.fname);
   printf("Enter the last name: ");
   gets(newname.lname);
   return newname;
}
```

- Call by Value (mainly)
- Returning the structure to the calling function
- Similar to returning a variable value in basic data type

 22

Passing by Returning Structures

The fouth approach is to return the structure in the function. The function returns a structure **nameTag**. To call this function, the calling function must declare a variable of type **struct nameTag** in order to receive the result from **getname()** as shown here which assigns the returned structure data to the variable **name**.



In summary, after watching this video lesson, you should be able to do the listed.