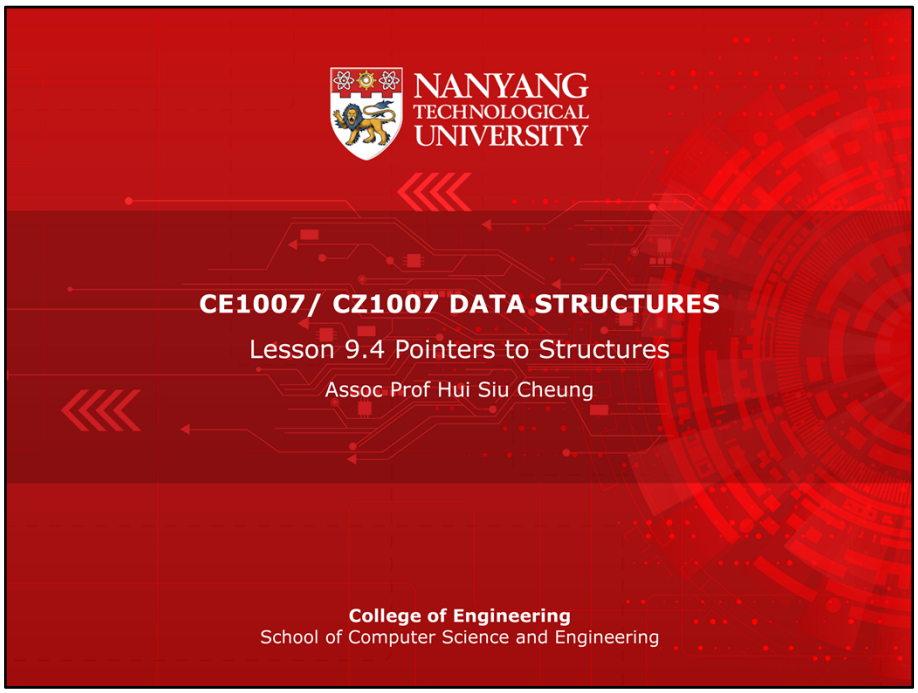
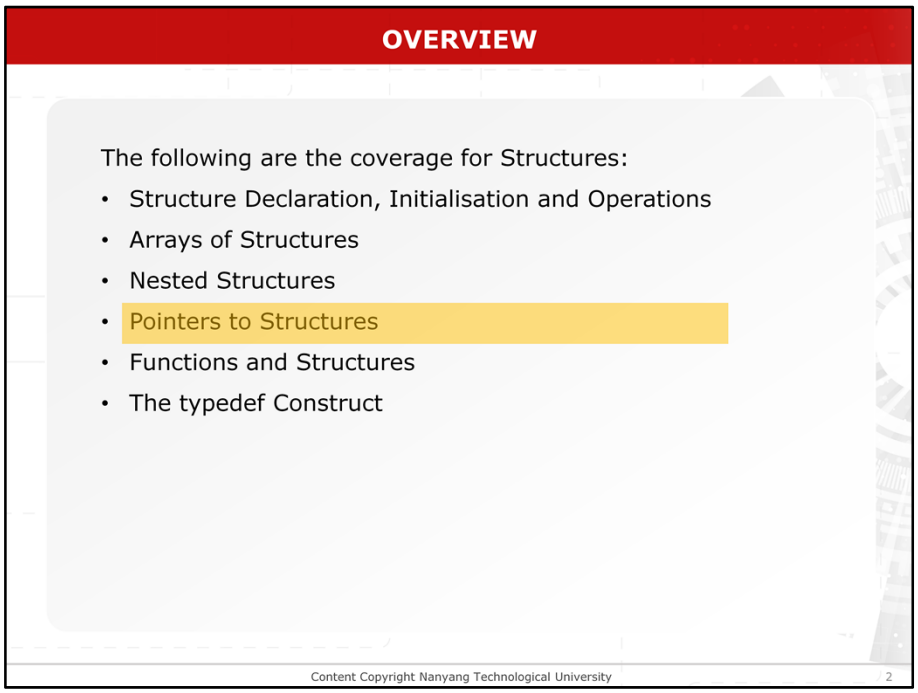
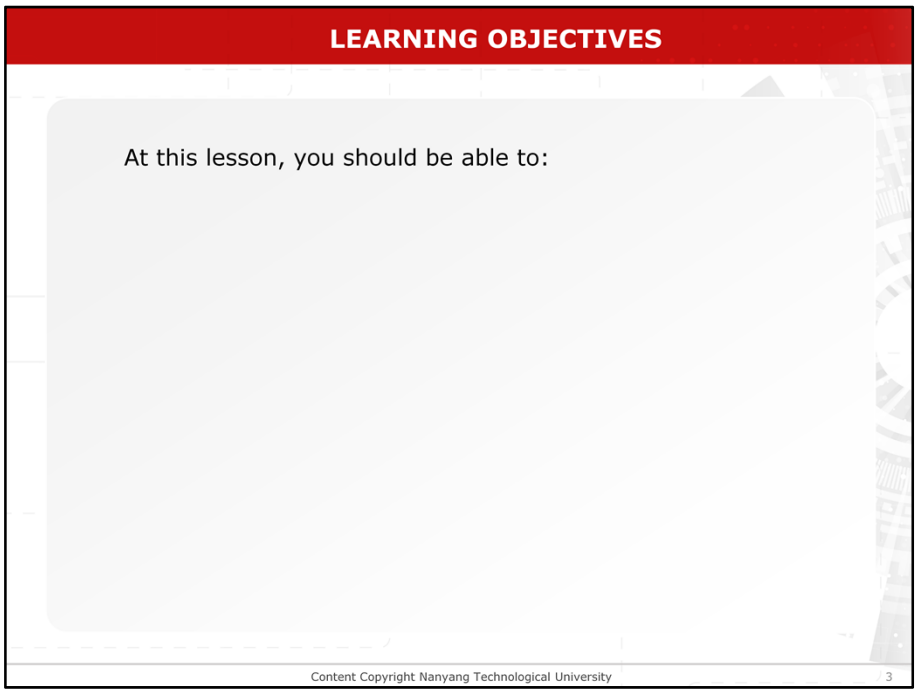
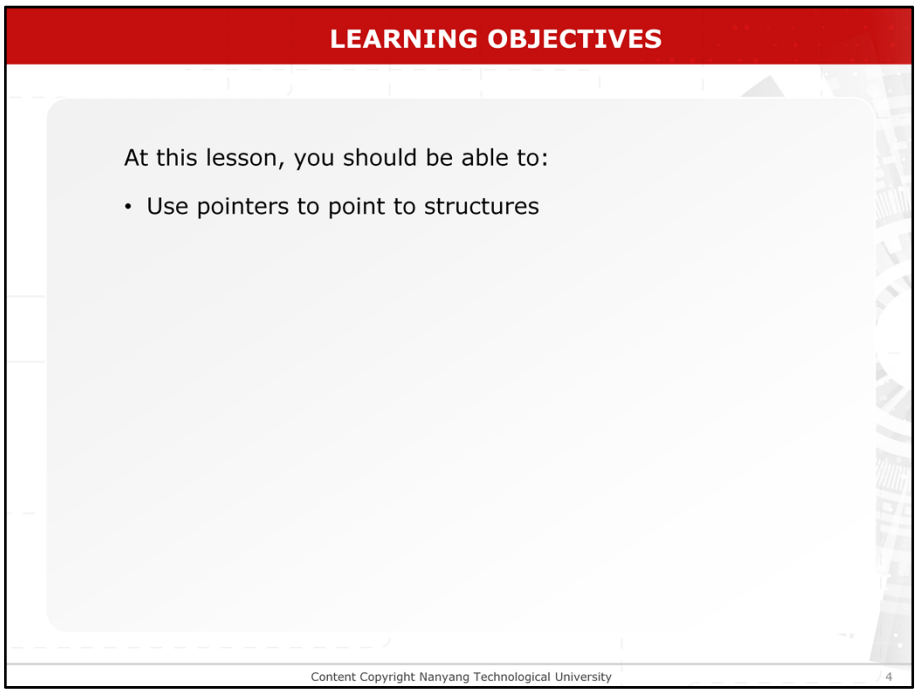

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 9.4 Pointers to Structures
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering



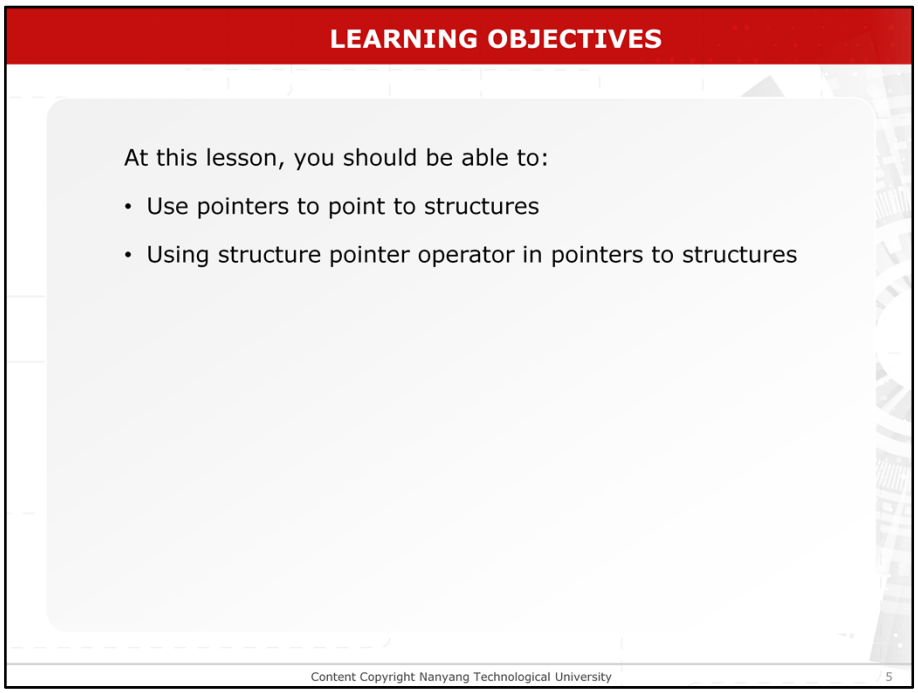
The following are the coverage for Structures: this video focusses on Pointers to Structures.



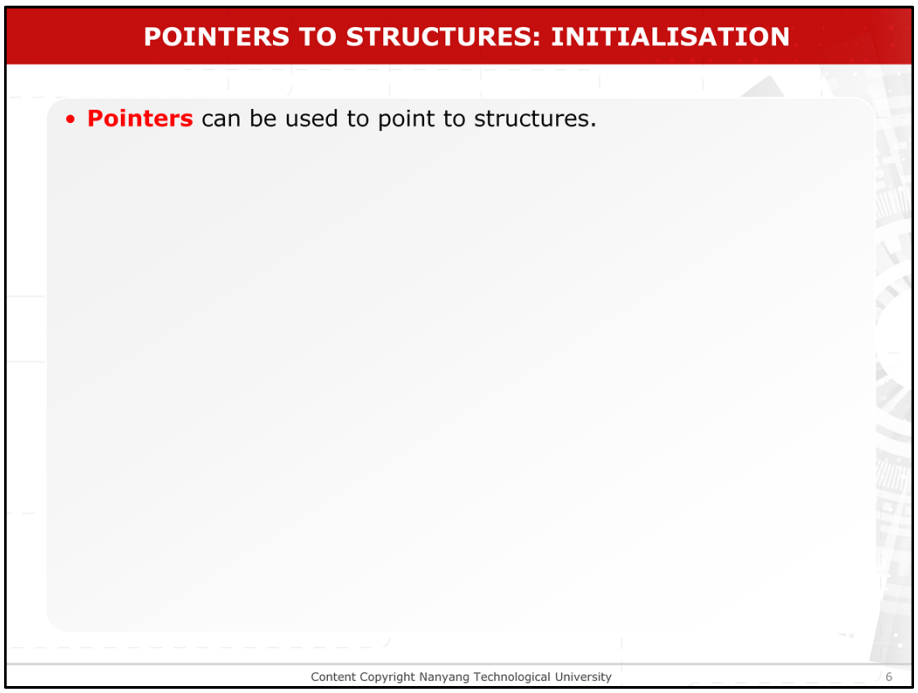
LEARNING OBJECTIVES: At this lesson, you should be able to:



Use pointers to point to structures

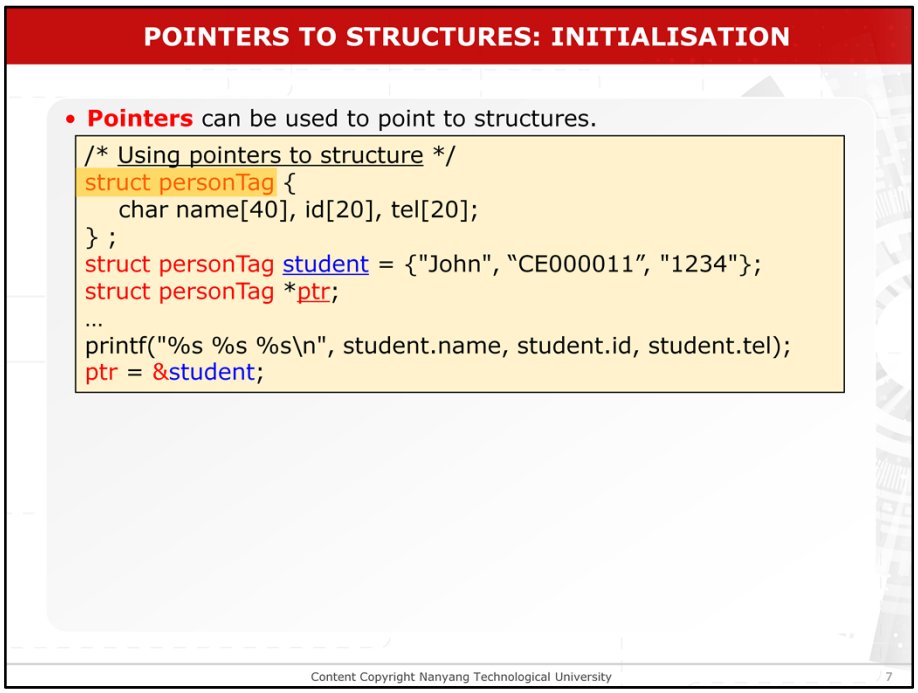


Using structure pointer operator in pointers to structures

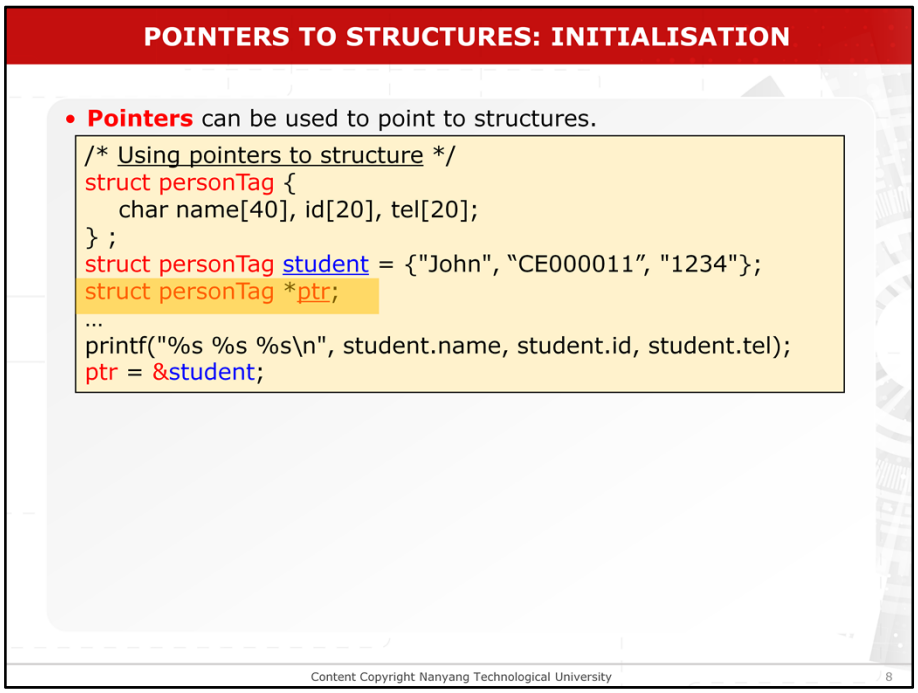


Pointers to Structures: Initialization

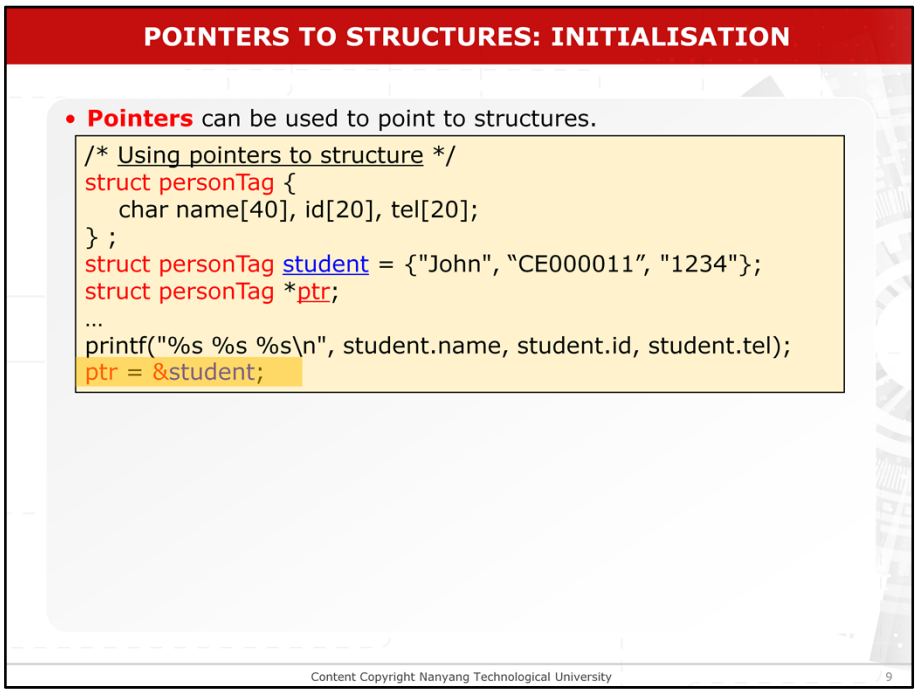
Pointers are flexible and powerful in C. They can be used to point to structures.



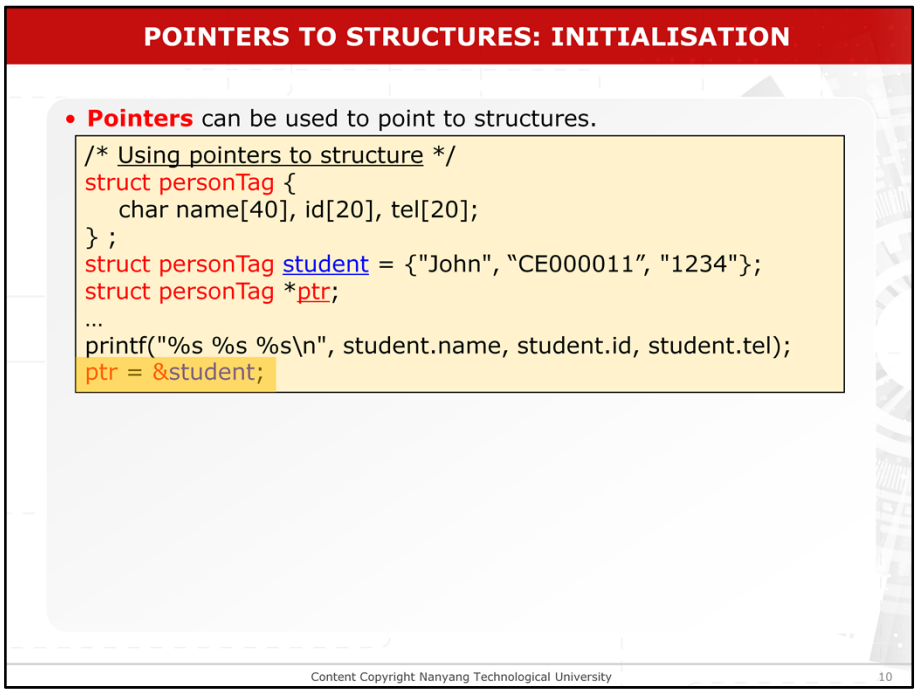
The declarations in the program shown here define a structure template **personTag**



and create a pointer **ptr** to the structure **personTag**.



To initialize a pointer, we must use the address operator (**&**) to obtain the address of a structure variable, and then assign the address to the pointer.

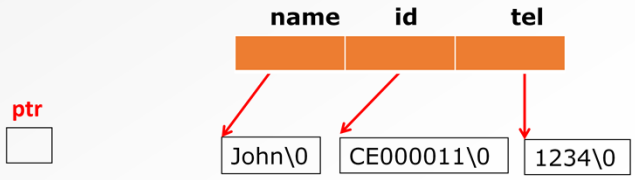


The address of the structure variable **student** is assigned to the pointer variable **ptr**. Then, we can use the pointer variable **ptr** to access the contents in the structure variable **student**.

POINTERS TO STRUCTURES: INITIALISATION

- **Pointers** can be used to point to structures.

```
/* Using pointers to structure */
struct personTag {
    char name[40], id[20], tel[20];
} ;
struct personTag student = {"John", "CE000011", "1234"};
struct personTag *ptr;
...
printf("%s %s %s\n", student.name, student.id, student.tel);
ptr = &student;
```

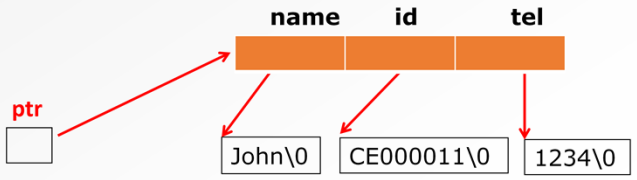


Then, we can use the pointer variable **ptr** to access the contents in the structure variable **student**.

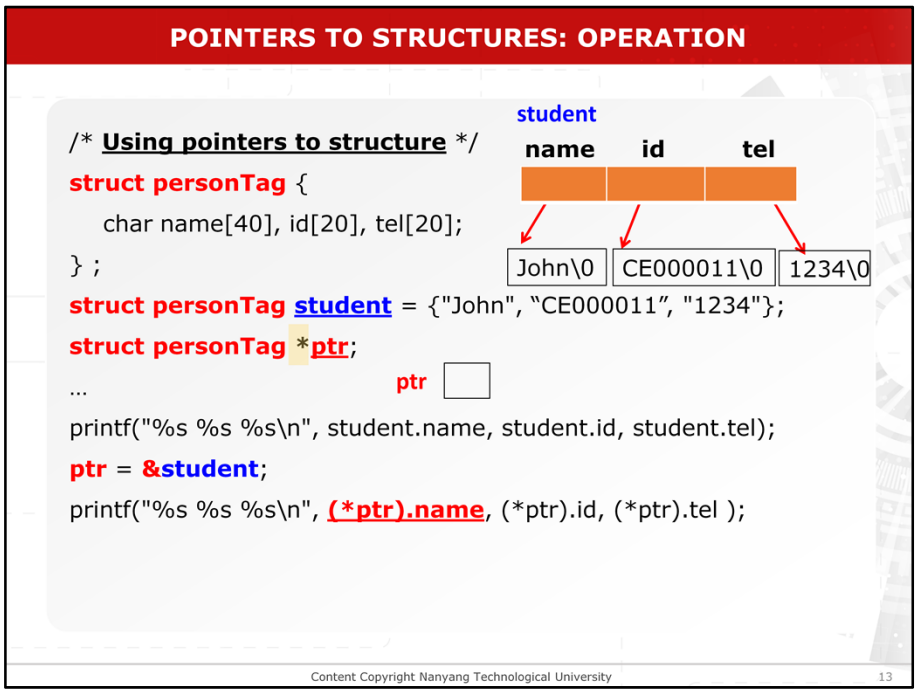
POINTERS TO STRUCTURES: INITIALISATION

- **Pointers** can be used to point to structures.

```
/* Using pointers to structure */
struct personTag {
    char name[40], id[20], tel[20];
} ;
struct personTag student = {"John", "CE000011", "1234"};
struct personTag *ptr;
...
printf("%s %s %s\n", student.name, student.id, student.tel);
ptr = &student;
```



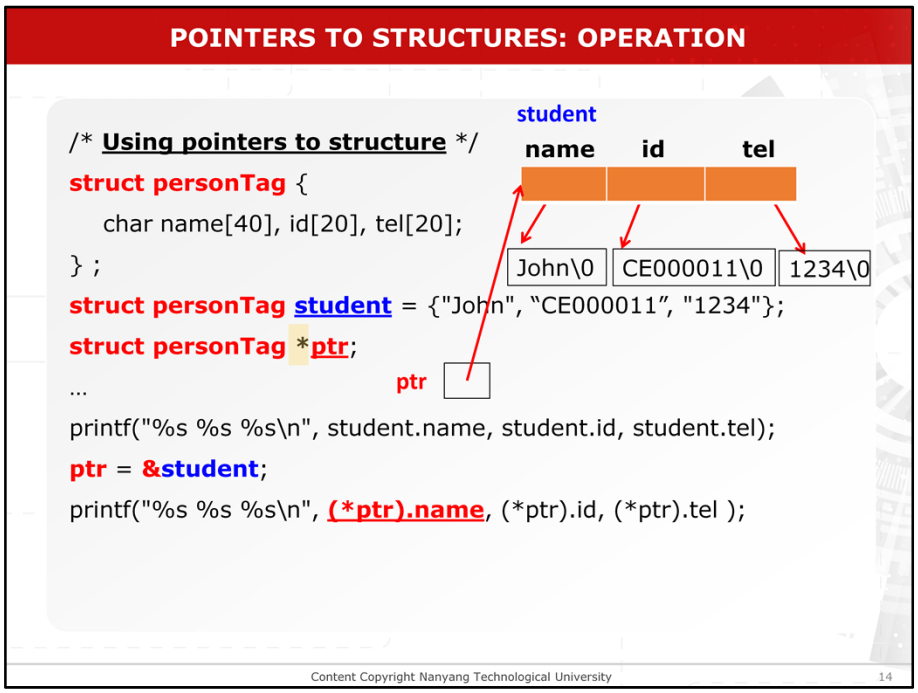
Then, we can use the pointer variable **ptr** to access the contents in the structure variable **student**.



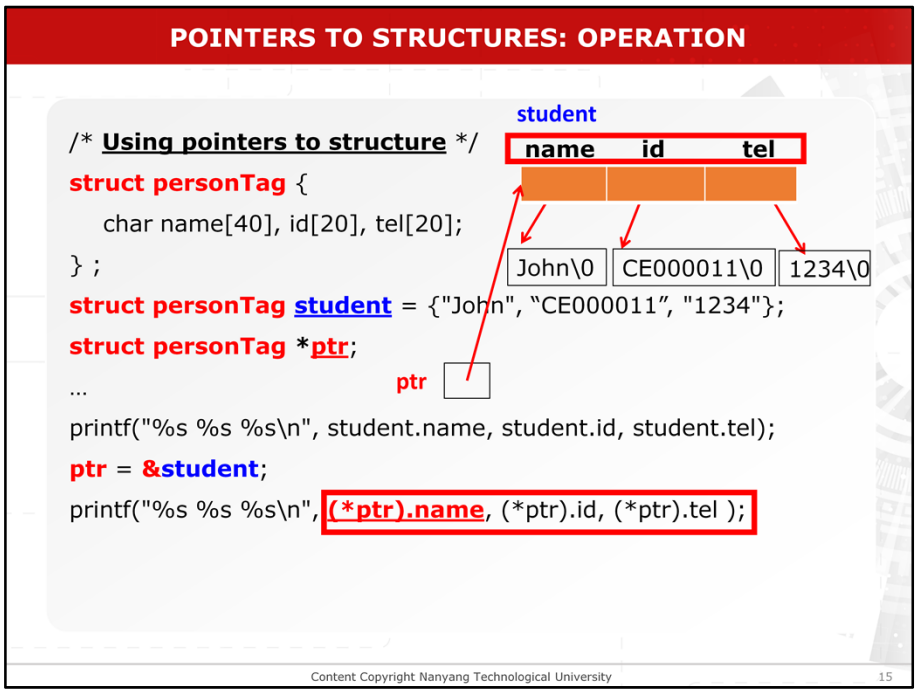
Pointers to Structures: Operation

The **indirection operator** (*****) can be used to access a member of a structure via a pointer to the structure.

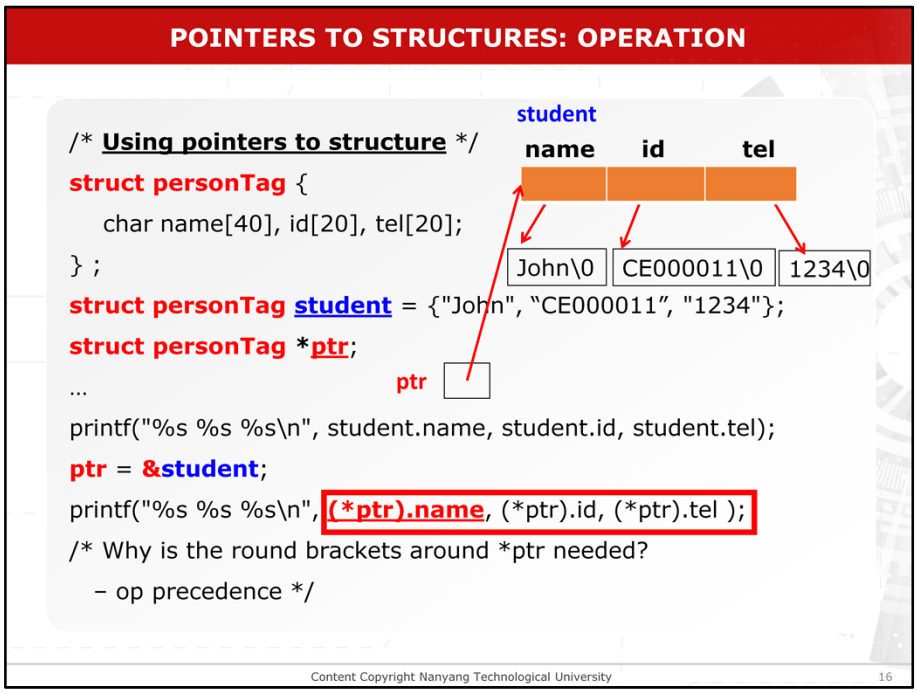
POINTERS TO STRUCTURES: OPERATION



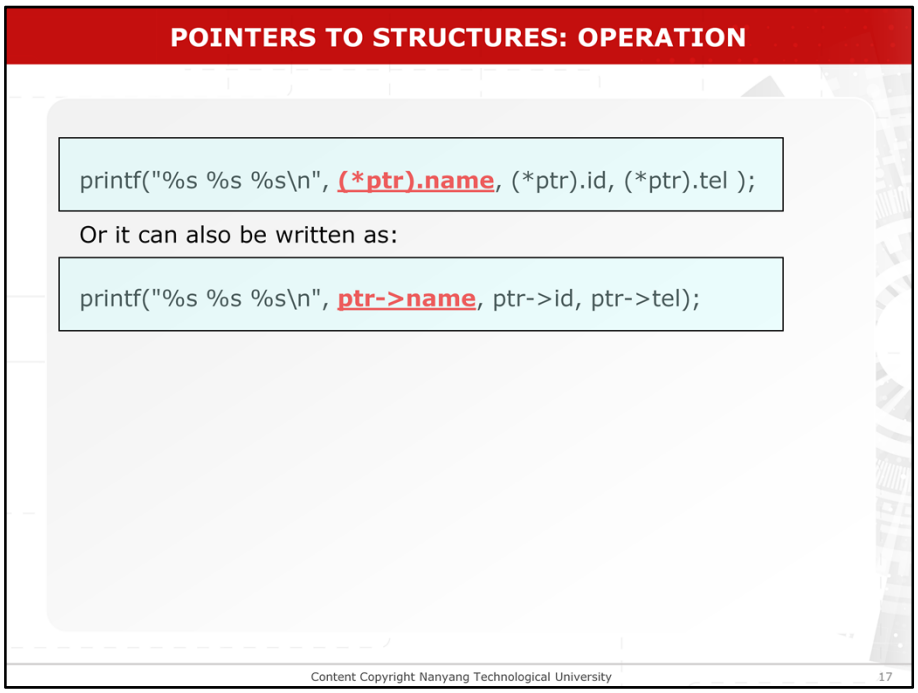
Since **ptr** points to the structure **student**



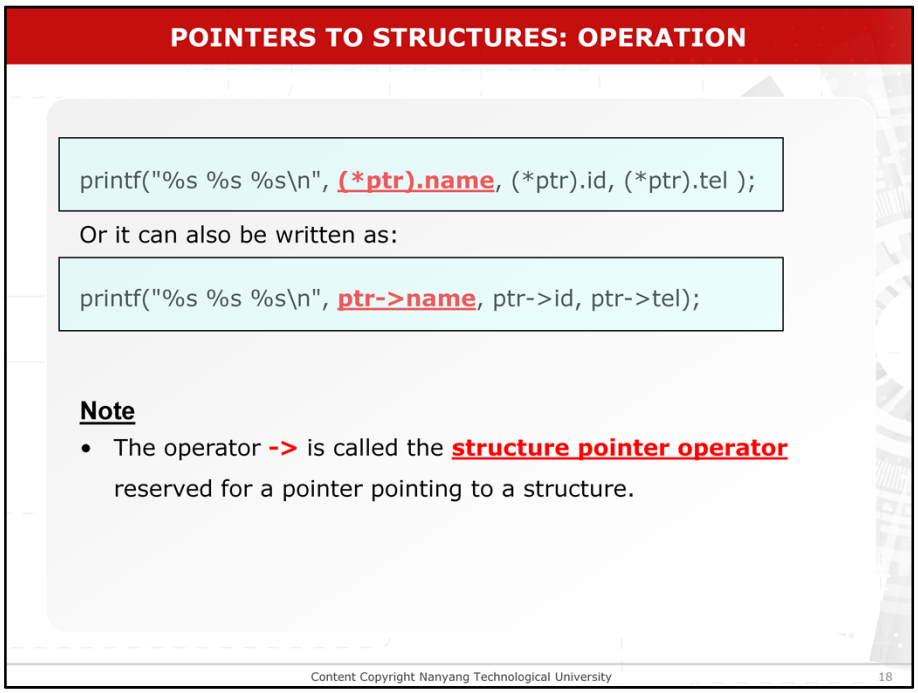
the notations shown in the red box return the value of the member **name**, **id** and **tel** of **student** respectively.



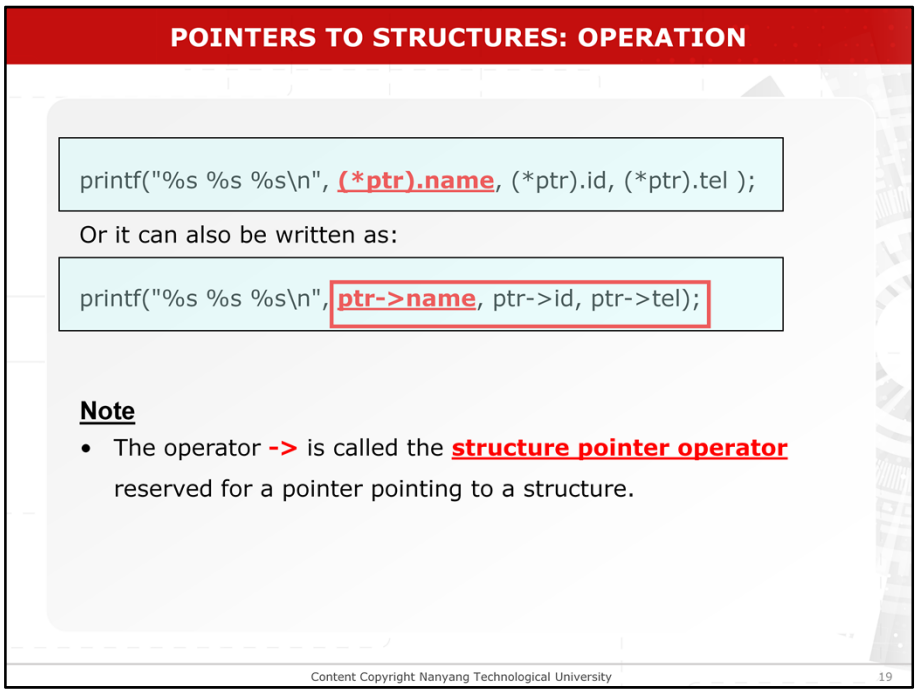
Why is the round brackets around pointer needed? The parentheses are necessary to enclose **pointer** as the member dot operator has higher precedence than the star which is the indirection operator



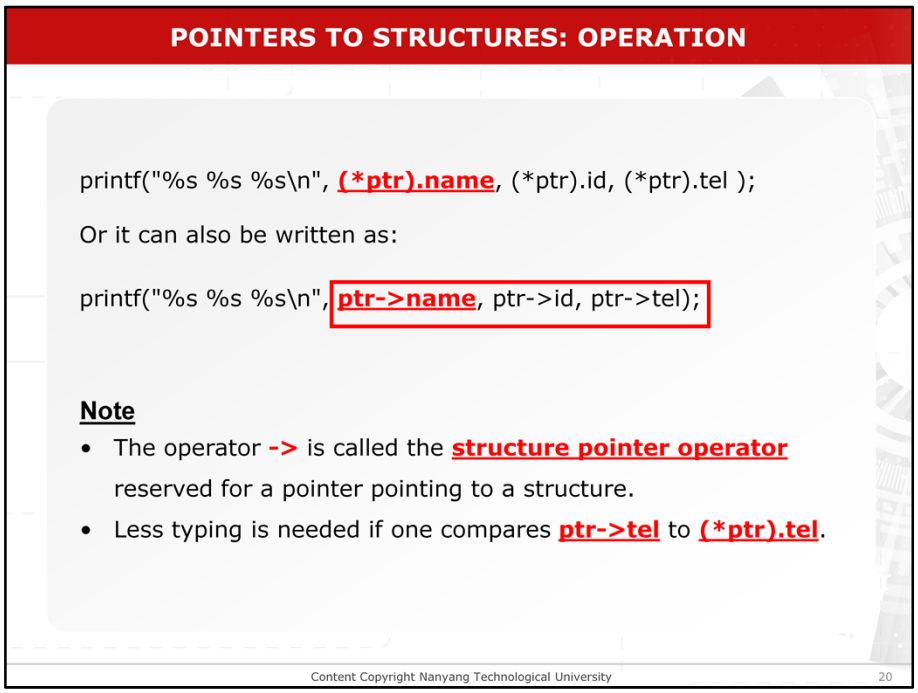
Since dereferencing is very common in pointer to structure



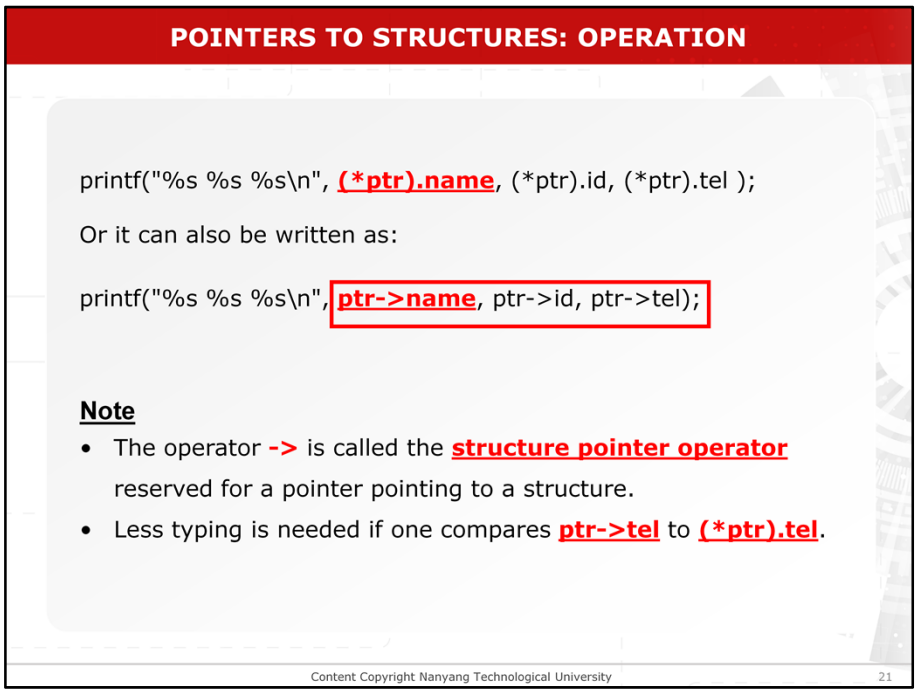
C provides an operator called the *structure pointer operator* for a pointer pointing to a structure. There is no whitespace between the symbols .



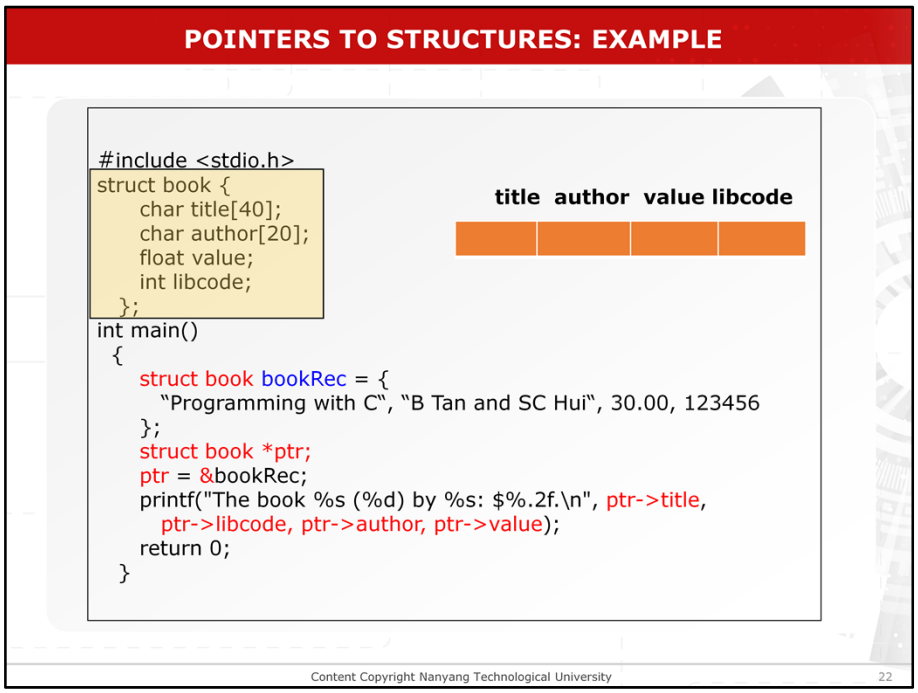
We can use the notations in the red box shown to obtain the values of the members of the structure **student**.



It takes less typing when using structure pointer operator though they have exactly the same meaning.

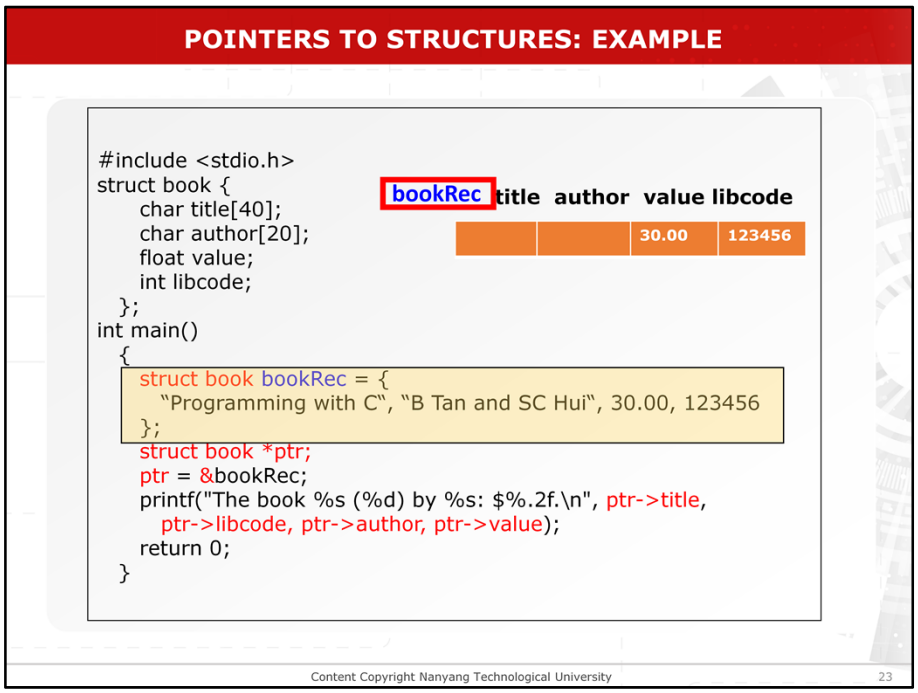


It is quite common to use the structure pointer operator (**->**) instead of the indirection operator (*****) in pointers to structures.



Pointers to Structures: Example

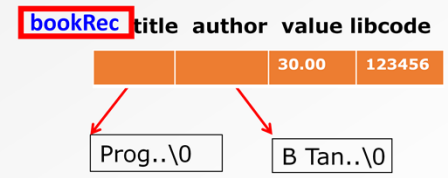
In the program, we define a structure called **book** with four members: **title**, **author**, **value** and **libcode**.



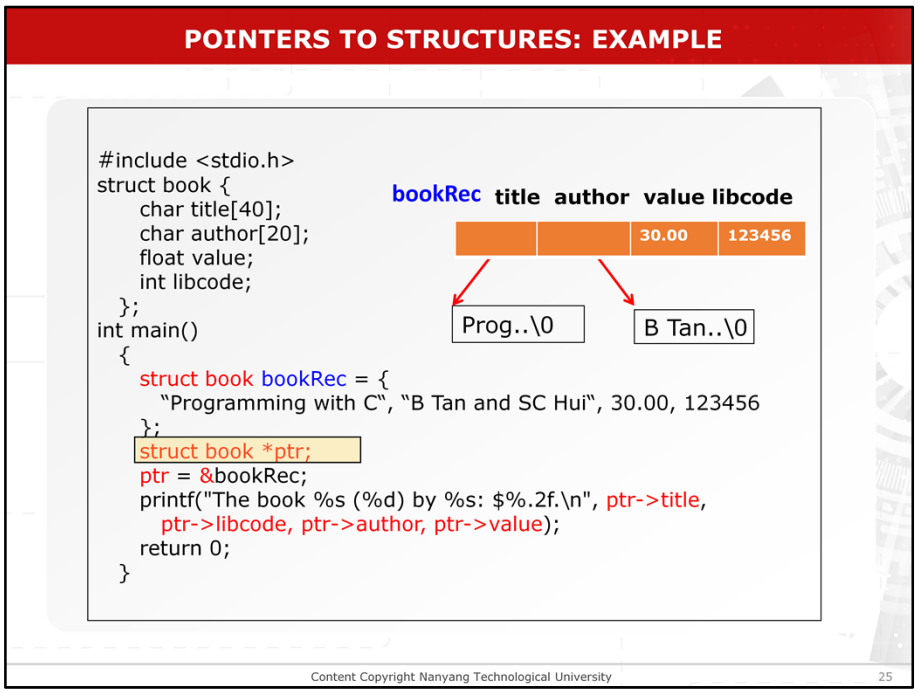
After that, we define a structure variable called **bookRec**

POINTERS TO STRUCTURES: EXAMPLE

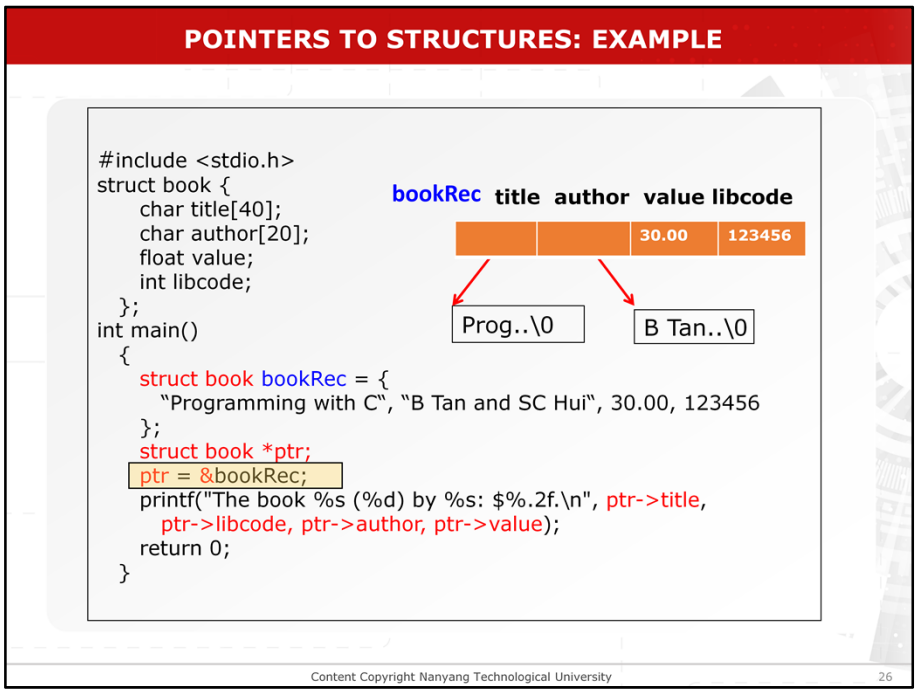
```
#include <stdio.h>
struct book {
     char title[40];
     char author[20];
     float value;
     int libcode;
  };
int main()
  {
    struct book bookRec = {
      "Programming with C", "B Tan and SC Hui", 30.00, 123456
    };
    struct book *ptr;
    ptr = &bookRec;
    printf("The book %s (%d) by %s: $%.2f.\n", ptr->title,
      ptr->libcode, ptr->author, ptr->value);
    return 0;
  }
```



and initialize it with values.



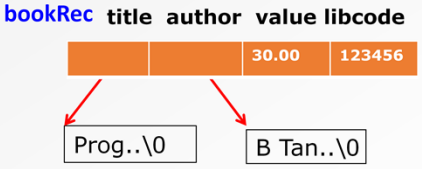
We then define the pointer variable **ptr** to the **struct book** type:



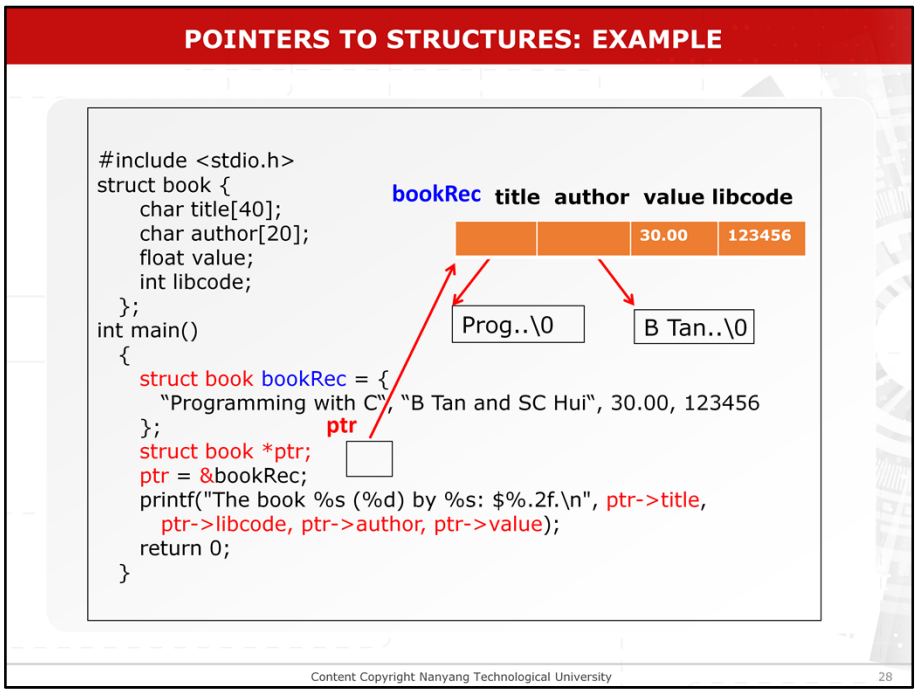
We assign the address of the structure variable **bookRec** to the pointer variable **ptr**:

POINTERS TO STRUCTURES: EXAMPLE

```
#include <stdio.h>
struct book {
     char title[40];
     char author[20];
     float value;
     int libcode;
  };
int main()
  {
     struct book bookRec = {
       "Programming with C", "B Tan and SC Hui", 30.00, 123456
     };
     struct book *ptr;
     ptr = &bookRec;
     printf("The book %s (%d) by %s: $%.2f.\n", ptr->title,
       ptr->libcode, ptr->author, ptr->value);
     return 0;
  }
```



Therefore, the pointer variable contains the address of **bookRec**.



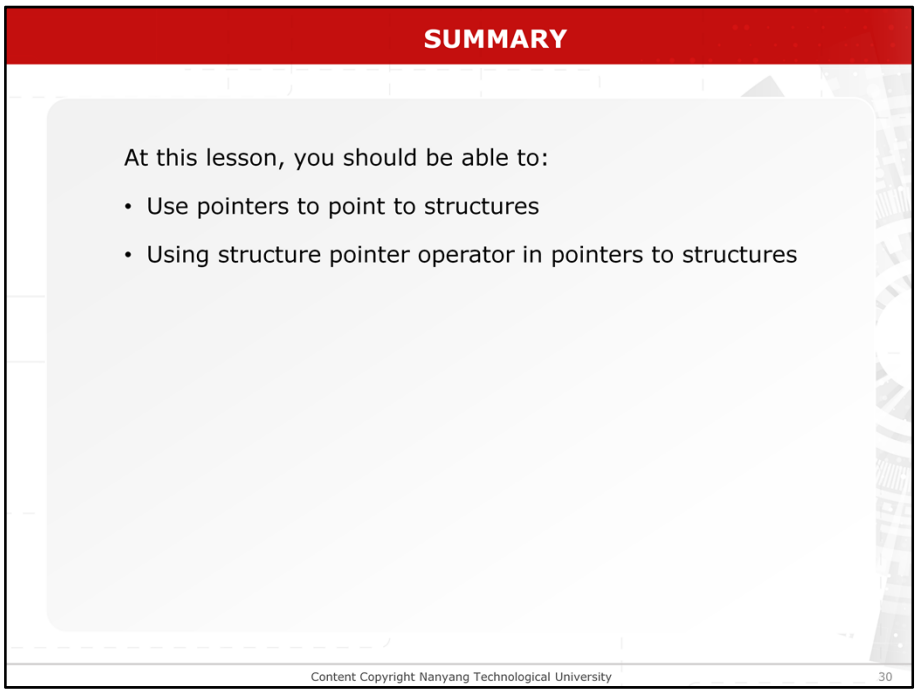
As a result, we may access the members of **bookRec** via **ptr**.

POINTERS TO STRUCTURES: EXAMPLE

```
#include <stdio.h>
struct book {
     char title[40];
     char author[20];
     float value;
     int libcode;
  };
int main()
  {
     struct book bookRec = {
       "Programming with C", "B Tan and SC Hui", 30.00, 123456
     };
     struct book *ptr;
     ptr = &bookRec;
     printf("The book %s (%d) by %s: $%.2f.\n", ptr->title,
       ptr->libcode, ptr->author, ptr->value);
     return 0;
  }
```

Output
The book Programming with C (123456) by B Tan and SC Hui: $30.00.

The statement shown prints each member information of **bookRec**.



After watching the video lesson, you will be able to do the listed.