

## OVERVIEW

The following are the coverage for Structures:

- Structure Declaration, Initialisation and Operations
- Arrays of Structures
- Nested Structures
- Pointers to Structures
- Functions and Structures
- The `typedef` Construct

Content Copyright Nanyang Technological University

2

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

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

Content Copyright Nanyang Technological University 3

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

## LEARNING OBJECTIVES

At this lesson, you should be able to:

- Use pointers to point to structures

Content Copyright Nanyang Technological University

4

Use pointers to point 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

Using structure pointer operator in pointers to structures

**POINTERS TO STRUCTURES: INITIALISATION**

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

Content Copyright Nanyang Technological University 6

### Pointers to Structures: Initialization

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

## 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;
```

Content Copyright Nanyang Technological University 7

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

## 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;
```

Content Copyright Nanyang Technological University 8

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

## 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;
```

Content Copyright Nanyang Technological University 9

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.

## 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;
```

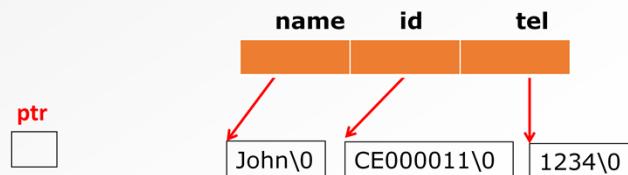
Content Copyright Nanyang Technological University 10

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;
```



Content Copyright Nanyang Technological University

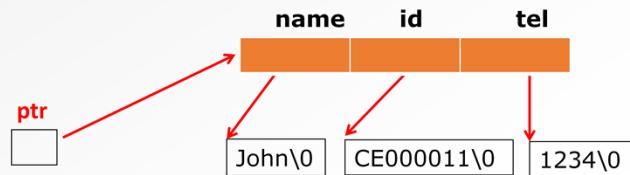
11

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;
```



Content Copyright Nanyang Technological University

12

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

## POINTERS TO STRUCTURES: OPERATION

```

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

```

**student**  

name	id	tel
John\0	CE000011\0	1234\0

Content Copyright Nanyang Technological University

13

### 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

```

/* 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;
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );

```

Content Copyright Nanyang Technological University

14

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

## POINTERS TO STRUCTURES: OPERATION

```

/* 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;
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );

```

**student**  

<b>name</b>	<b>id</b>	<b>tel</b>
John\0	CE000011\0	1234\0

Content Copyright Nanyang Technological University

15

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

## POINTERS TO STRUCTURES: OPERATION

```

/* 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;
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );
/* Why is the round brackets around *ptr needed?
   - op precedence */

```

Content Copyright Nanyang Technological University      16

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**

## POINTERS TO STRUCTURES: OPERATION

```
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );
```

Or it can also be written as:

```
printf("%s %s %s\n", ptr->name, ptr->id, ptr->tel);
```

Since dereferencing is very common in pointer to structure

## POINTERS TO STRUCTURES: OPERATION

```
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );
```

Or it can also be written as:

```
printf("%s %s %s\n", ptr->name, ptr->id, ptr->tel);
```

### Note

- The operator **->** is called the **structure pointer operator** reserved for a pointer pointing to a structure.

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

## POINTERS TO STRUCTURES: OPERATION

```
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );
```

Or it can also be written as:

```
printf("%s %s %s\n", ptr->name, ptr->id, ptr->tel);
```

### Note

- The operator `->` is called the **structure pointer operator** reserved for a pointer pointing to a structure.

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

## POINTERS TO STRUCTURES: OPERATION

```
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );
```

Or it can also be written as:

```
printf("%s %s %s\n", ptr->name, ptr->id, ptr->tel);
```

### Note

- The operator `->` is called the **structure pointer operator** reserved for a pointer pointing to a structure.
- Less typing is needed if one compares `ptr->tel` to `(*ptr).tel`.

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

## POINTERS TO STRUCTURES: OPERATION

```
printf("%s %s %s\n", (*ptr).name, (*ptr).id, (*ptr).tel );
```

Or it can also be written as:

```
printf("%s %s %s\n", ptr->name, ptr->id, ptr->tel);
```

### Note

- The operator `->` is called the **structure pointer operator** reserved for a pointer pointing to a structure.
- Less typing is needed if one compares `ptr->tel` to `(*ptr).tel`.

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

## 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;
}
```

title	author	value	libcode

Content Copyright Nanyang Technological University 22

### Pointers to Structures: Example

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

## 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;
}
```

Content Copyright Nanyang Technological University 23

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;
}
```

bookRec title author value libcode

		30.00	123456
	Prog..\\0	B Tan..\\0	

Content Copyright Nanyang Technological University 24

and initialize it with values.

## 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;
}
```

**bookRec title author value libcode**

			30.00	123456
Prog..\\0	B Tan..\\0			

Content Copyright Nanyang Technological University

25

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

## 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;
}
```

**bookRec**    **title**    **author**    **value**    **libcode**

		30.00	123456
Prog..\\0	B Tan..\\0		

Content Copyright Nanyang Technological University

26

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;
}
```

bookRec	title	author	value	libcode
			30.00	123456
	Prog..\\0	B Tan..\\0		

Content Copyright Nanyang Technological University

27

Therefore, the pointer variable contains the address of **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;
}
```

**bookRec**    **title**    **author**    **value**    **libcode**

			30.00	123456
Prog..\\0	B Tan..\\0			

Content Copyright Nanyang Technological University

28

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.

Content Copyright Nanyang Technological University 29

The `printf` statement shown prints each member information of `bookRec`.

## SUMMARY

At this lesson, you should be able to:

- Use pointers to point to structures
- Using structure pointer operator in pointers to structures

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