

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**

**CE1007/ CZ1007 DATA STRUCTURES**

Lesson 9.1 Structure Declaration, Initialisation & Operations

Assoc Prof Hui Siu Cheung

**College of Engineering**  
School of Computer Science and Engineering

## REVIEW

**What have you learnt so far:**

Content Copyright Nanyang Technological University

2

**From the previous lessons, you have learnt the following:**

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)

Content Copyright Nanyang Technological University

3

Basic Sequential Programming. We covered data, operators, simple Input, Output.

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)
- Branching (if-else, switch, conditional operator)

Content Copyright Nanyang Technological University

4

Branching using if-else, switch, conditional operator.

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)
- Branching (if-else, switch, conditional operator)
- Looping (for, while, do-while)

Content Copyright Nanyang Technological University

5

Looping using for, while, and do, while

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)
- Branching (if-else, switch, conditional operator)
- Looping (for, while, do-while)
- Functions (function definition and call by value)

Content Copyright Nanyang Technological University

6

Functions where we learnt about function definition and call by value.

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)
- Branching (if-else, switch, conditional operator)
- Looping (for, while, do-while)
- Functions (function definition and call by value)
- Pointers (call by reference for function communication)

Content Copyright Nanyang Technological University

7

Pointers where we learnt about call by reference for function communication.

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)
- Branching (if-else, switch, conditional operator)
- Looping (for, while, do-while)
- Functions (function definition and call by value)
- Pointers (call by reference for function communication)
- 1D arrays (using pointers for 1D arrays)

Content Copyright Nanyang Technological University

8

## 1 Dimensional arrays using pointers

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)
- Branching (if-else, switch, conditional operator)
- Looping (for, while, do-while)
- Functions (function definition and call by value)
- Pointers (call by reference for function communication)
- 1D arrays (using pointers for 1D arrays)
- 2D (Multidimensional) arrays (using pointers for 2D arrays)

Content Copyright Nanyang Technological University

9

## 2 dimensional arrays using pointers

## REVIEW

### What have you learnt so far:

- Basic Sequential Programming (data, operators, simple I/O)
- Branching (if-else, switch, conditional operator)
- Looping (for, while, do-while)
- Functions (function definition and call by value)
- Pointers (call by reference for function communication)
- 1D arrays (using pointers for 1D arrays)
- 2D (Multidimensional) arrays (using pointers for 2D arrays)
- Character strings (array of characters with null character)

Content Copyright Nanyang Technological University

10

Character strings (array of characters with null character)

## REVIEW

- Arrays are lists of data of the same data type.

Content Copyright Nanyang Technological University

11

Arrays are lists of data of the same data type.

Character strings are arrays of characters.

If you need to process data of different types as a record (structure), C provides you with Structures.

In this lecture, we discuss about ... **Structures**....

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 15

## Structures

The following are the coverage for structures. This video is focused on the 1<sup>st</sup> topic: structure declaration, initialization and operations

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

Content Copyright Nanyang Technological University 16

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

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

- Define structure template

Content Copyright Nanyang Technological University 17

Define structure template

## LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define structure template
- Declare structure variables

### Declare structure variables

## LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define structure template
- Declare structure variables
- Initialise structure variables

Content Copyright Nanyang Technological University 19

### Initialise structure variables

## LEARNING OBJECTIVES

At this lesson, you should be able to:

- Define structure template
- Declare structure variables
- Initialise structure variables
- Assign structure variables using assignment operators

Assign structure variables using assignment operators

## RECORDS

- Medical Records



Content Copyright Nanyang Technological University 21

### Records

Records are used to keep related information of an object together. There are many examples of records such as medical records

Employee records

book records, etc. Structure is similar to record in that it is used to keep related data together as a data type.

After defining a structure, we can then define a variable of that structure to store the different data together under that variable.

The slide has a red header bar with the word 'STRUCTURES' in white capital letters. Below the header is a light gray content area containing a single bullet point:

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.

At the bottom of the slide, there is a footer bar with the text 'Content Copyright Nanyang Technological University' on the left and the number '25' on the right.

### Structures

Structure is an aggregate of values. Their components are distinct, and may possibly have different types, including arrays and other structures. To build our own data types using structures, we need to define the structure and declare variables of that type. A structure template is used to specify a structure definition. It tells the compiler the various components that make up the structure. Structure variables are then declared with the type of the structure.

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:



Content Copyright Nanyang Technological University 26

### Structures

For example, a **record** about a book in a library may contain,

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:
  - **char title[40];**



Content Copyright Nanyang Technological University 27

may contain the title

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:
  - **char** title[40];
  - **char** author[20];



Content Copyright Nanyang Technological University

28

author

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:
  - **char** title[40];
  - **char** author[20];
  - **float** value;



Content Copyright Nanyang Technological University

29

and book value.

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:
  - **char** title[40];
  - **char** author[20];
  - **float** value;

[Note: may have **different** data types]



Content Copyright Nanyang Technological University

30

A structure can then be defined as a **data type** with the different data members.

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:
  - **char** title[40];
  - **char** author[20];
  - **float** value;

[Note: may have **different** data types]

- Two steps to use a structure:

Content Copyright Nanyang Technological University

31



To use a structure in a program, there are two steps:

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:
  - **char** title[40];
  - **char** author[20];
  - **float** value;

[Note: may have **different** data types]



- Two steps to use a structure:
  - Setting up a structure template (similar to a data type);

Content Copyright Nanyang Technological University

32

### Structures

Set up a structure template similar to a data type);

## STRUCTURES

- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.
- For example, a **record** about a book in a library may contain, i.e. book record:
  - **char** title[40];
  - **char** author[20];
  - **float** value;

[Note: may have **different** data types]



Content Copyright Nanyang Technological University

33

defining a variable based on the structure template

**DEFINING A STRUCTURE TEMPLATE**

- A **structure template** is the master plan that describes how a structure is put together. To set up a structure template, e.g.

Content Copyright Nanyang Technological University 34

### Structure Template (Data Type)

A structure template (or data type) is the master plan that describes how a structure is put together. A structure template can be set up as follows:

## DEFINING A STRUCTURE TEMPLATE

- A **structure template** is the master plan that describes how a structure is put together. To set up a structure template, e.g.

```
struct book {           /* template of book */  
    char title[40];  
    char author[20];   /* members */  
    float value;       /* semicolon to end the definition */  
};
```

Content Copyright Nanyang Technological University

35

### Structure Template (Data Type)

A structure template can be seen below

## DEFINING A STRUCTURE TEMPLATE

- A **structure template** is the master plan that describes how a structure is put together. To set up a structure template, e.g.

```
struct book {           /* template of book */  
    char title[40];  
    char author[20];   /* members */  
    float value;       /* semicolon to end the definition */  
};
```

- **struct**: reserved keyword to introduce a structure

Content Copyright Nanyang Technological University

36

**struct** is a reserved keyword to introduce a structure.

## DEFINING A STRUCTURE TEMPLATE

- A **structure template** is the master plan that describes how a structure is put together. To set up a structure template, e.g.

```
struct book {           /* template of book */  
    char title[40];  
    char author[20];   /* members */  
    float value;  
};                      /* semicolon to end the definition */
```

- **struct**: reserved keyword to introduce a structure
- **book**: an **optional tag name** which follows the keyword **struct** to name the structure declared.

Content Copyright Nanyang Technological University

37

**book** is an optional tag name that follows the keyword **struct** to name the structure declared.

## DEFINING A STRUCTURE TEMPLATE

- A **structure template** is the master plan that describes how a structure is put together. To set up a structure template, e.g.

```
struct book {           /* template of book */  
    char title[40];  
    char author[20];   /* members */  
    float value;  
};                      /* semicolon to end the definition */
```

- **struct**: reserved keyword to introduce a structure
- **book**: an **optional tag name** which follows the keyword struct to name the structure declared.
- title, author, value: the **member** of the structure book.

Content Copyright Nanyang Technological University

38

The **title**, **author** and **value** are the *members* of the structure **book**. The members of a structure can be any of the valid C data types. A semicolon after the closing brace ends the definition of the structure definition.

## DEFINING A STRUCTURE TEMPLATE

- A **structure template** is the master plan that describes how a structure is put together. To set up a structure template, e.g.

```
struct book {           /* template of book */  
    char title[40];  
    char author[20];   /* members */  
    float value;  
};                      /* semicolon to end the definition */
```

- **struct**: reserved keyword to introduce a structure
- **book**: an **optional tag name** which follows the keyword struct to name the structure declared.
- title, author, value: the **member** of the structure book.
- Note - The above declaration just declares a template, not a variable. **No memory space** is allocated.

Content Copyright Nanyang Technological University

39

The declaration declares a template (or data type), not a variable. Therefore, no memory space is allocated. It only acts as a template for the named structure type. The tag name **book** can then be used for the declaration of variables.

## DECLARING STRUCTURE VARIABLE: WITH TAG NAME

- **With tag name:** separate the definition of structure template from the definition of structure variable.

Content Copyright Nanyang Technological University

40

### Structure Variable: with Tag Name

The structure name or tag is optional. With structure tag, the definition of structure template can be separated from the definition of structure variables.

## DECLARING STRUCTURE VARIABLE: WITH TAG NAME

- With tag name: separate the definition of structure template from the definition of structure variable.

```
struct person {  
    char name[20];  
    int age;  
    float salary;  
};  
struct person tom, mary;
```

name	age	salary
ptr	int	float

Array of 20 chars

Content Copyright Nanyang Technological University

41

In the declaration shown here, a structure template **person** comprising three components is created.

## DECLARING STRUCTURE VARIABLE: WITH TAG NAME

- With tag name: separate the definition of structure template from the definition of structure variable.

```
struct person {  
    char name[20];  
    int age;  
    float salary;  
};  
struct person tom, mary;
```

name	age	salary
ptr	int	float

↓

Array of 20 chars

Content Copyright Nanyang Technological University

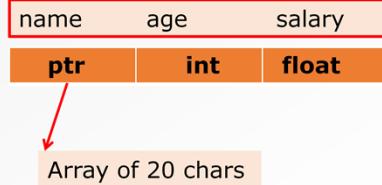
42

**tom** and **mary** are two structure variables which are declared using the structure **person**. With tag name, we can use the structure data type subsequently in the program.

## DECLARING STRUCTURE VARIABLE: WITH TAG NAME

- **With tag name:** separate the definition of structure template from the definition of structure variable.

```
struct person {  
    char name[20];  
    int age;  
    float salary;  
};  
struct person tom, mary;
```



- With tag name – we can use the structure type subsequently in the program.

Content Copyright Nanyang Technological University

43

With tag name, we can use the structure data type subsequently in the program.

## DECLARING STRUCTURE VARIABLE: WITHOUT TAG NAME

- **Without tag name:** combine the definition of structure template with that of structure variable.

### Declaring Structure Variable without Tag Name

Without structure tag, the definition of structure template must be combined with that of structure variables.

## DECLARING STRUCTURE VARIABLE: WITHOUT TAG NAME

- **Without tag name:** combine the definition of structure template with that of structure variable.

```
struct {  
    char name[20];  
    int age;  
    float salary;  
} tom, mary; /* no tag – person is not used */  
/* structure variables */
```

Content Copyright Nanyang Technological University

45

In the declaration shown, a structure template is created with the three components:

## DECLARING STRUCTURE VARIABLE: WITHOUT TAG NAME

- **Without tag name:** combine the definition of structure template with that of structure variable.

```
struct {  
    char name[20];  
    int age;  
    float salary;  
} tom, mary; /* no tag – person is not used */  
/* structure variables */
```

Content Copyright Nanyang Technological University

46

The variables **tom** and **mary** are then defined using this structure.

Without structure tag name, we cannot use the structure elsewhere in the program. It is always a good idea to include a structure tag when defining a structure.

## DECLARING STRUCTURE VARIABLE: WITHOUT TAG NAME

- **Without tag name:** combine the definition of structure template with that of structure variable.

```
struct {  
    char name[20];  
    int age;  
    float salary;  
} tom, mary; /* no tag – person is not used */  
/* structure variables */
```

- Without tag name – we cannot use the structure type elsewhere in the program.

Content Copyright Nanyang Technological University

47

Without structure tag name, we cannot use the structure elsewhere in the program. It is always a good idea to include a structure tag when defining a structure.

The slide has a red header bar with the title "ACCESSING STRUCTURE MEMBERS". Below the header is a white content area containing a bullet point and a code example. The bullet point states: "The notation required to reference the members of a structure is". Below this, a code example is shown in a red-bordered box: "structureVariableName.memberName". At the bottom of the slide, there is a footer bar with the text "Content Copyright Nanyang Technological University" and the number "48".

### Accessing Structure Members

The notation required to access a member of a structure is shown here

## ACCESSING STRUCTURE MEMBERS

- The notation required to reference the members of a structure is  
**structureVariableName.memberName**
- For example, to access the member age of the variable tom, we use  
**tom.age**

Content Copyright Nanyang Technological University 49

For example, to access the member **age** of the variable **tom**, we use **tom dot age**

## ACCESSING STRUCTURE MEMBERS

- The notation required to reference the members of a structure is  
**structureVariableName.memberName**
- For example, to access the member age of the variable tom, we use  
**tom.age**
- The "." (dot notation) is called ***member access operator***.

Content Copyright Nanyang Technological University 50

The dot notation is called *member access operator*. The member operator has the highest or equal priority among the operators.

## STRUCTURE DECLARATION & OPERATION: EXAMPLE

```

/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
}
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: $%.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
/* to access each member, we use the . notation */

```

### Output

Please enter the book title:

*C Programming*

Please enter the author:

*SC Hui*

Please enter the value:

*10.00*

C Programming by SC Hui: \$10.00

Content Copyright Nanyang Technological University
51

### Structure Declaration and Operation: Example

In the program, it defines the structure template (data type) and the declaration of a structure variable.

**STRUCTURE DECLARATION & OPERATION: EXAMPLE**

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: %.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

Content Copyright Nanyang Technological University

52

After defining the structure template **struct book** outside the **main** function, the following declaration

**struct book bookRec;**

declares a variable **bookRec** of type **struct book**. It also allocates storage for the variable.

**STRUCTURE DECLARATION & OPERATION: EXAMPLE**

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: $%.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

Content Copyright Nanyang Technological University

53

The structure definition can be placed inside a function or outside a function. If it is defined inside the function, the definition can only be used by that function. In the program, the definition is defined at the beginning of the file, it is a global declaration, and all the functions following the definition can use the template.

**STRUCTURE DECLARATION & OPERATION: EXAMPLE**

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: %.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

Content Copyright Nanyang Technological University

54

In the program, the statements HIGHLIGHTED HERE will read the user input on title and author which are character strings.

### STRUCTURE DECLARATION & OPERATION: EXAMPLE

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: %.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

Content Copyright Nanyang Technological University

55

To access a member of a structure, we use the dot notation such as **bookRec dot title** and **bookRec dot author**.

**STRUCTURE DECLARATION & OPERATION: EXAMPLE**

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: $%.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

Content Copyright Nanyang Technological University

56

The `scanf` statement will read the user input on book value which is of data type **float**.

**STRUCTURE DECLARATION & OPERATION: EXAMPLE**

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title); /* to access member, using . notation */
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: %.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

**Output**

Please enter the book title:  
C Programming

Content Copyright Nanyang Technological University

57

After reading the user input, the book title,

**STRUCTURE DECLARATION & OPERATION: EXAMPLE**

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: %.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

**Output**

Please enter the book title:

*C Programming*

Please enter the author:

*SC Hui*

Content Copyright Nanyang Technological University

58

author

**STRUCTURE DECLARATION & OPERATION: EXAMPLE**

```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
}
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: %.2f\n", bookRec.title, bookRec.author, bookRec.value);
    return 0;
}
```

**Output**

Please enter the book title:

*C Programming*

Please enter the author:

*SC Hui*

Please enter the value:

10.00

Content Copyright Nanyang Technological University

59

and book value will then be printed on the screen.

## STRUCTURE DECLARATION & OPERATION: EXAMPLE

```

/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    printf("Now enter the value:\n");
    scanf("%f", &bookRec.value); /* note that & is needed here */
    printf("%s by %s: $%.2f\n", bookRec.title, bookRec.author, bookRec.value)
    return 0;
}

```

**Output**

Please enter the book title:

*C Programming*

Please enter the author:

*SC Hui*

Please enter the value:

*10.00*

C Programming by SC Hui: \$10.00

**Q: What is the memory size of the variable bookRec?**

Content Copyright Nanyang Technological University
60

In the structure definition **struct book**, what is the memory size of the variable **bookRec** after declaring the variable under the data type **struct book**? You may use the **sizeof** operator to print the size of the variable.

## STRUCTURE VARIABLE INITIALISATION

- Syntax for **initialising structure variable** is **similar to** that for initializing array variable.

Content Copyright Nanyang Technological University

61

### Structure Variable Initialization and Operation

The syntax for initializing structure variable is similar to that of initializing array variable

## STRUCTURE VARIABLE INITIALISATION

- Syntax for **initialising structure variable** is **similar to** that for initializing array variable.
- When there are **insufficient** values assigned to all members of the structure, remaining members are assigned with **zero** by default.

Content Copyright Nanyang Technological University

62

When there are insufficient values to be assigned to all members of the structure, the remaining members are assigned to zero by default.

## STRUCTURE VARIABLE INITIALISATION

- Syntax for **initialising structure variable** is **similar to** that for initializing array variable.
- When there are **insufficient** values assigned to all members of the structure, remaining members are assigned with **zero** by default.

```
struct personTag{  
    char    name[20];  
    char    id[20];  
    char    tel[20];  
} student = {"John", "123", "456"};  
printf("%s %s %s\n", student.name, student.id, student.tel);
```

Content Copyright Nanyang Technological University

63

- . The structure variable is followed by an assignment symbol and a list of values defined within braces:

## STRUCTURE VARIABLE INITIALISATION

```

struct personTag{
    char name[20];
    char id[20];
    char tel[20];
} student = {"John", "123", "456"};
printf("%s %s %s\n", student.name, student.id, student.tel);

```

<b>name</b>	<b>id</b>	<b>tel</b>
John\0	123\0	456\0

Content Copyright Nanyang Technological University

64

Initialization of variables can only be performed with constant values or constant expressions that deliver a value of the required type. The initial values are assigned to the individual members of the structure in the order in which the members occur.

## STRUCTURE VARIABLE INITIALISATION

```
struct personTag{  
    char    name[20];  
    char    id[20];  
    char    tel[20];  
} student = {"John", "123", "456"};  
printf("%s %s %s\n", student.name, student.id, student.tel);
```

Content Copyright Nanyang Technological University 65

The **name** member of **student** is assigned with "John", the **id** member is assigned with "123", and the **tel** member is assigned with "456".

## STRUCTURE VARIABLE INITIALISATION

```

struct personTag{
    char name[20];
    char id[20];
    char tel[20];
} student = {"John", "123", "456"};
printf("%s %s %s\n", student.name, student.id, student.tel);

```

**using . notation**

<b>name</b>	<b>id</b>	<b>tel</b>
John\0	123\0	456\0

**Output**  
John 123 456

Content Copyright Nanyang Technological University

66

To print the data of the structure variable **student**, the highlighted statement and the dot notation is used in order to access the member of the structure.

**STRUCTURE ASSIGNMENT**

- The values in one structure can be assigned to another:

```
struct personTag newmember;
```

Content Copyright Nanyang Technological University 67

### Structure Assignment

The value of one structure variable can be assigned to another structure variable of the same type using the assignment operator. First, we define a new variable **newmember** under the data type **struct personTag**:

The slide has a red header bar with the title "STRUCTURE ASSIGNMENT". Below the header, there is a list bullet point: "• The values in one structure can be assigned to another:". Underneath this, there is a code snippet:

```
struct personTag newmember;  
newmember = student;
```

At the bottom of the slide, there is a footer bar with the text "Content Copyright Nanyang Technological University" and the number "68".

Then, we can assign the **struct personTag** variable **student** to **newmember** as shown;

## STRUCTURE ASSIGNMENT

- The values in one structure can be assigned to another:

```
struct personTag newmember;
```

```
newmember = student;
```

- This has the effect of copying the entire contents of the structure variable **student** to the structure variable **newmember**.

This has the effect of copying the entire contents of the structure variable **student** to the structure variable **newmember**.

## STRUCTURE ASSIGNMENT

- The values in one structure can be assigned to another:

```
struct personTag newmember;
```

```
    newmember = student;
```

- This has the effect of copying the entire contents of the structure variable **student** to the structure variable **newmember**.
- Each member of the **newmember** variable is assigned with the value of the corresponding member in the **student** variable.

Content Copyright Nanyang Technological University

70

Each member of the **newmember** variable is assigned with the value of the corresponding member in the **student** variable.

**SUMMARY**

At this lesson, you should be able to:

- Define structure template
- Declare structure variable
- Initialise structure variables
- Assign structure variables using assignment operators

In summary, after viewing this video lesson, your should be able to do the listed.