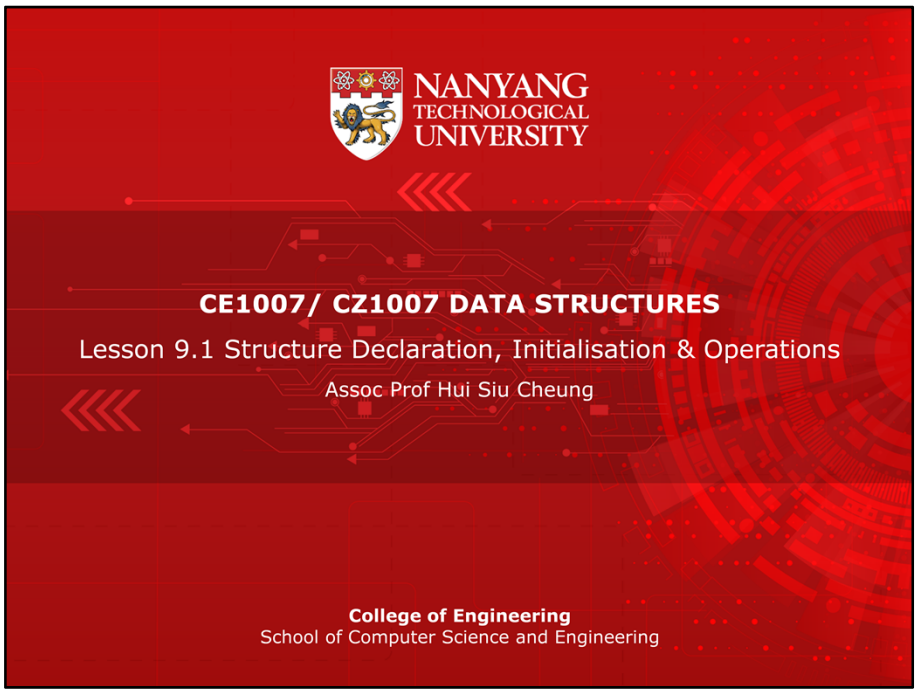
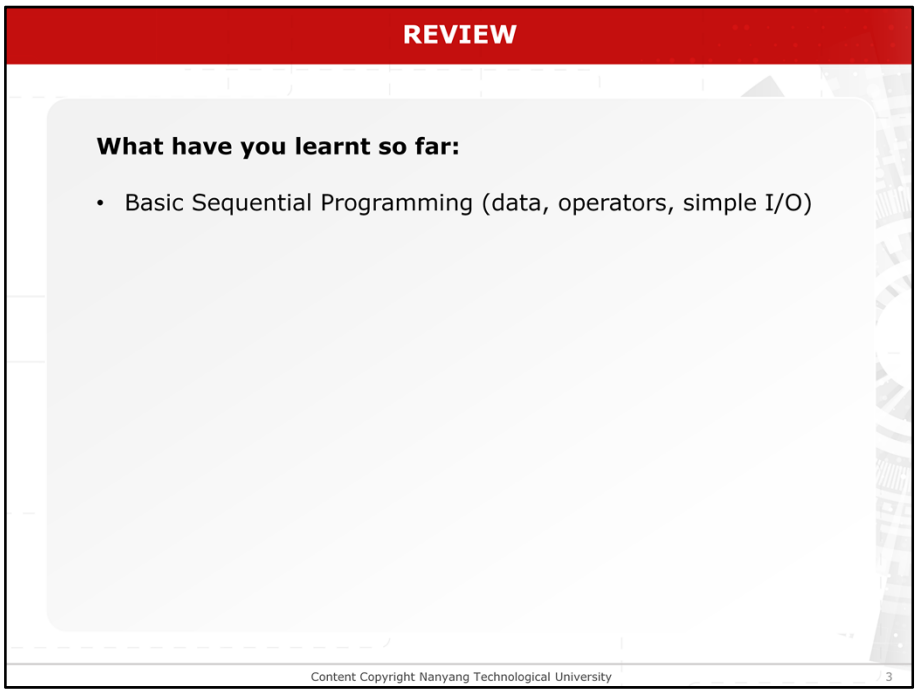
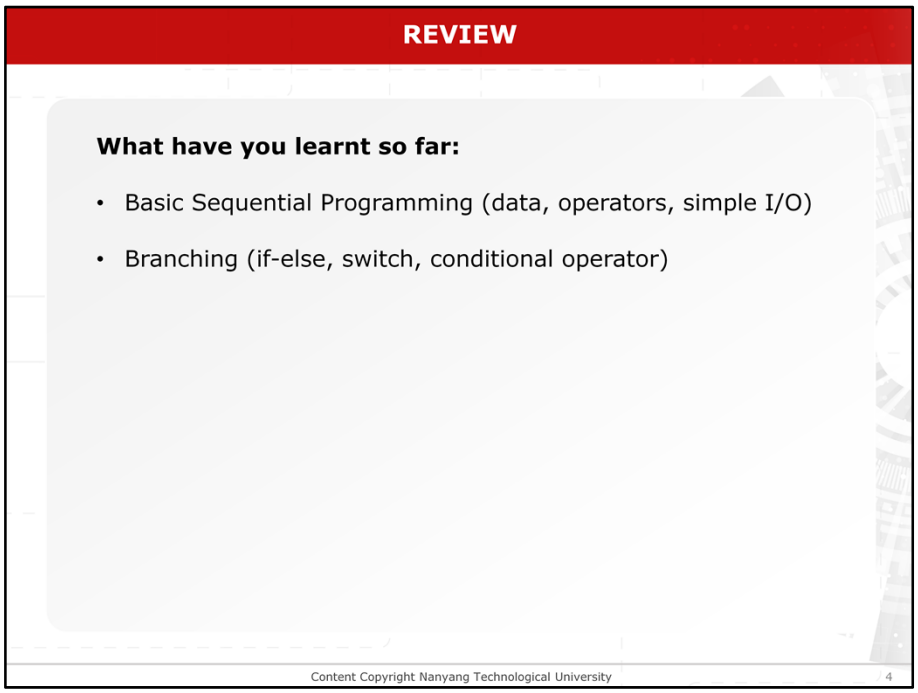

NANYANG
TECHNOLOGICAL
UNIVERSITY
CE1007/ CZ1007 DATA STRUCTURES
Lesson 9.1 Structure Declaration, Initialisation & Operations
Assoc Prof Hui Siu Cheung
College of Engineering
School of Computer Science and Engineering

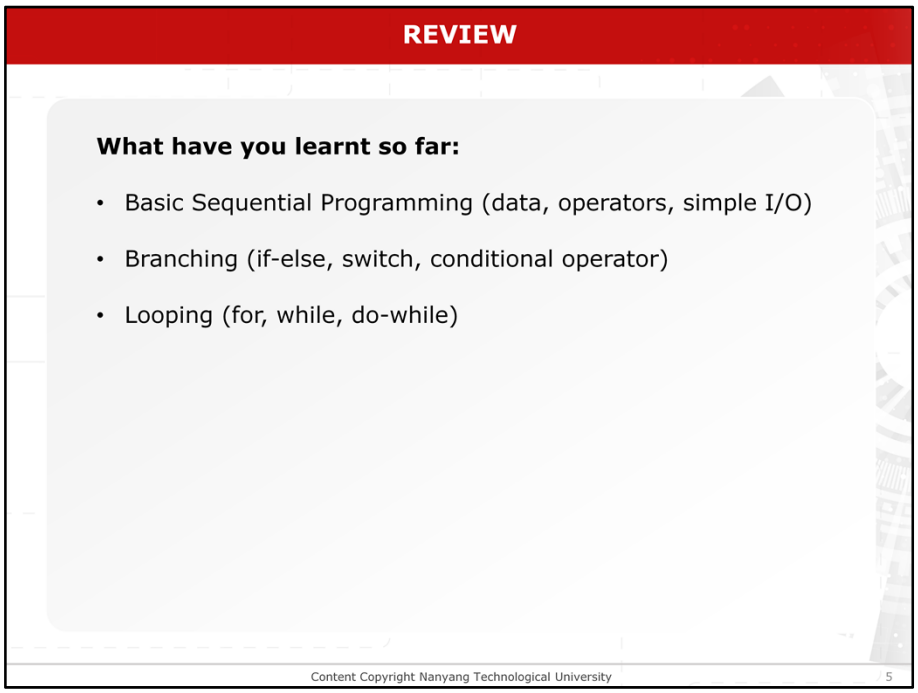
From the previous lessons, you have learnt the following:



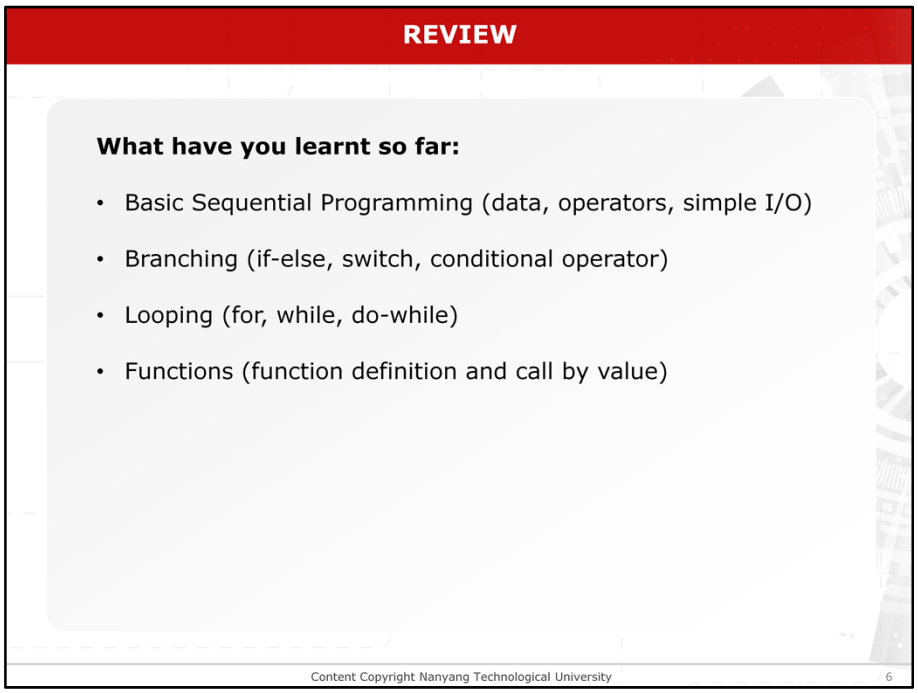
Basic Sequential Programming. We covered data, operators, simple Input, Output.



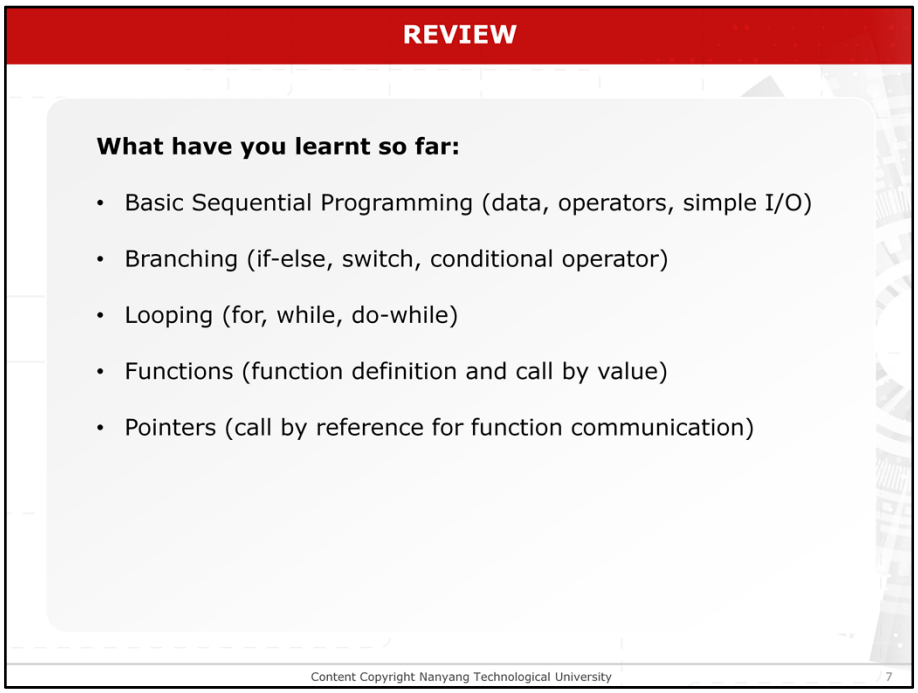
Branching using if-else, switch, conditional operator.



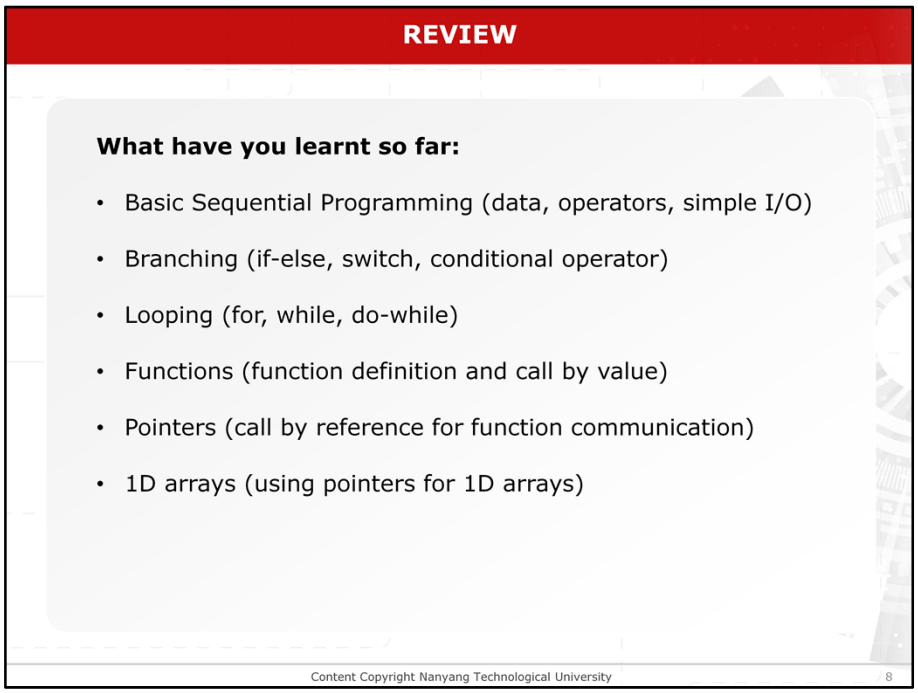
Looping using for, while, and do, while



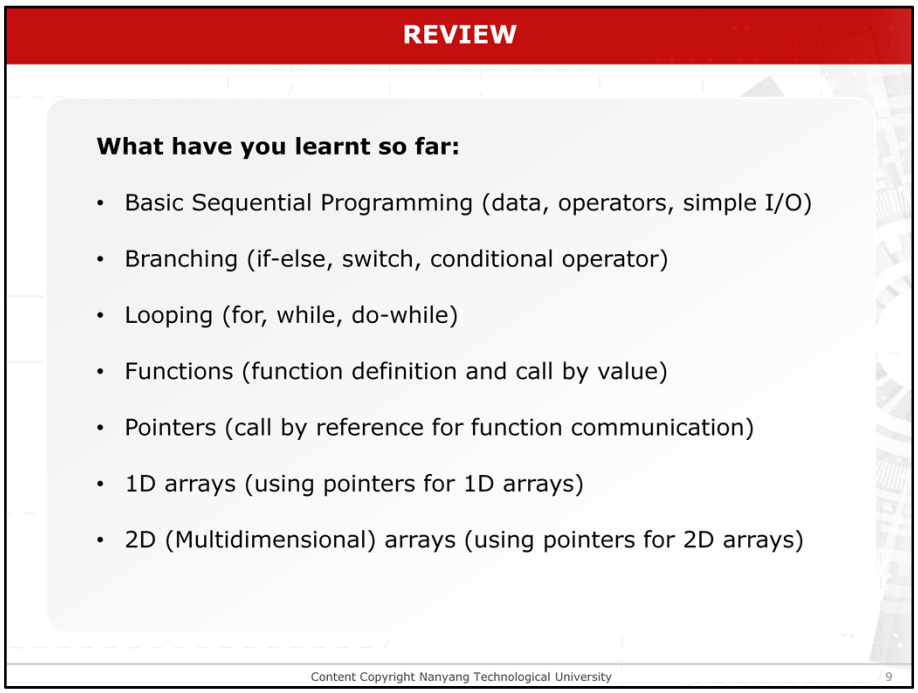
Functions where we learnt about function definition and call by value.



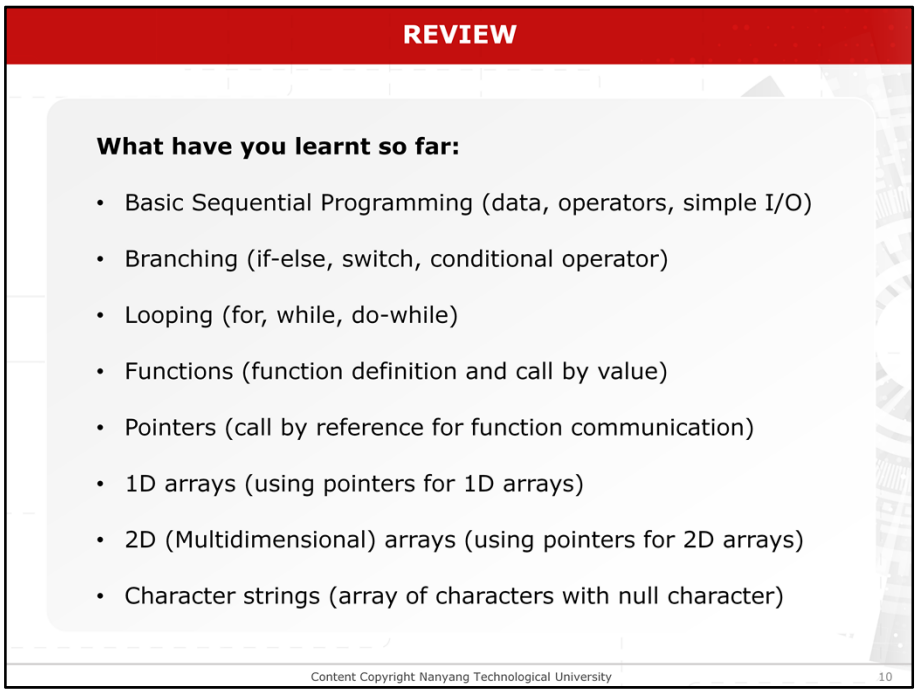
Pointers where we learnt about call by reference for function communication.



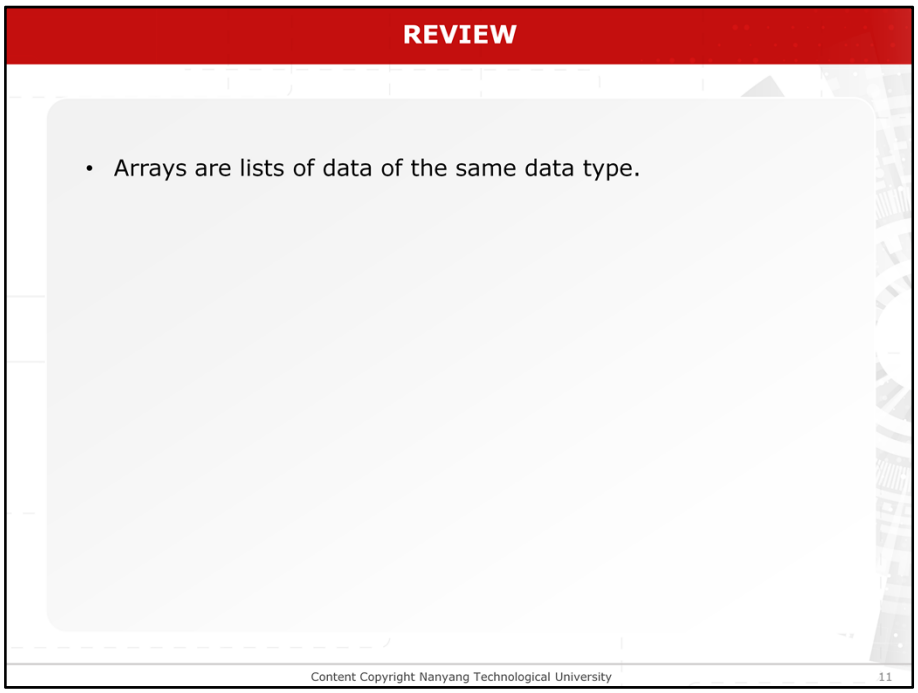
1 Dimensional arrays using pointers



2 dimensional arrays using pointers



Character strings (array of characters with null character)

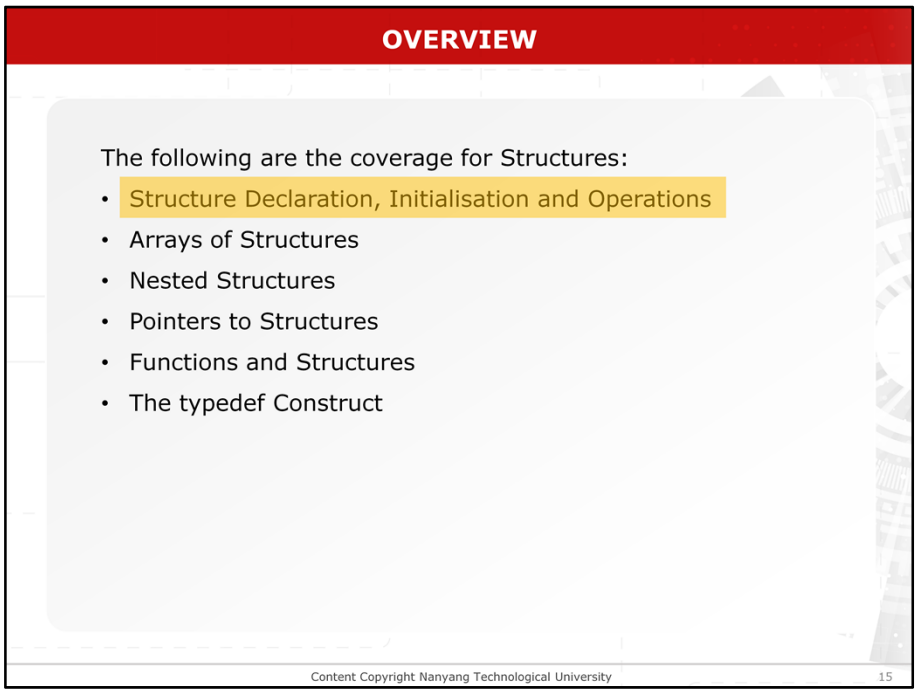


Arrays are lists of data of the same data type.

Character strings are arrays of characters.

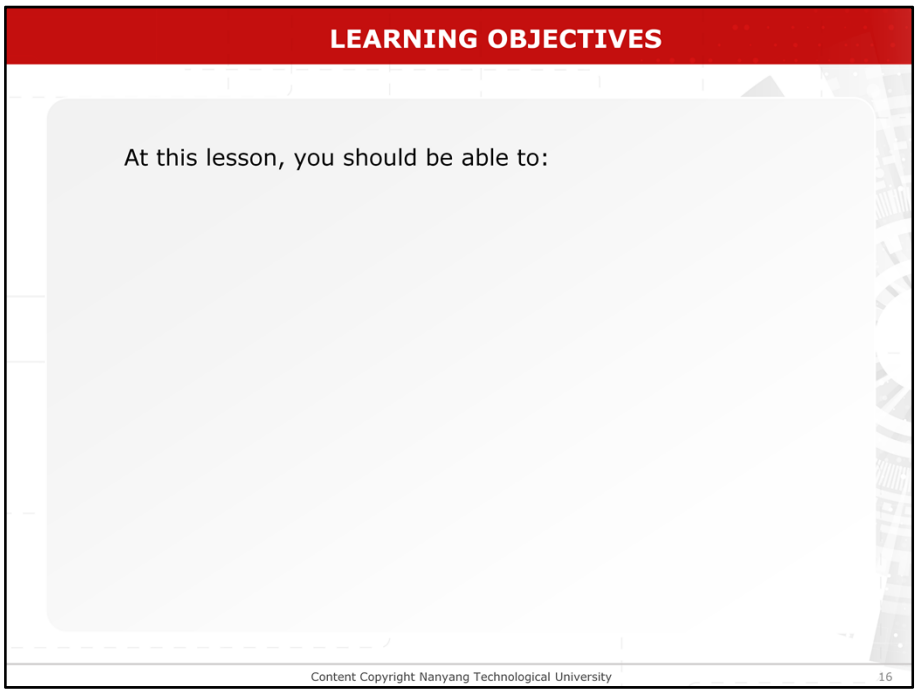
If you need to process data of different types as a record (structure), C provides you with Structures.

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

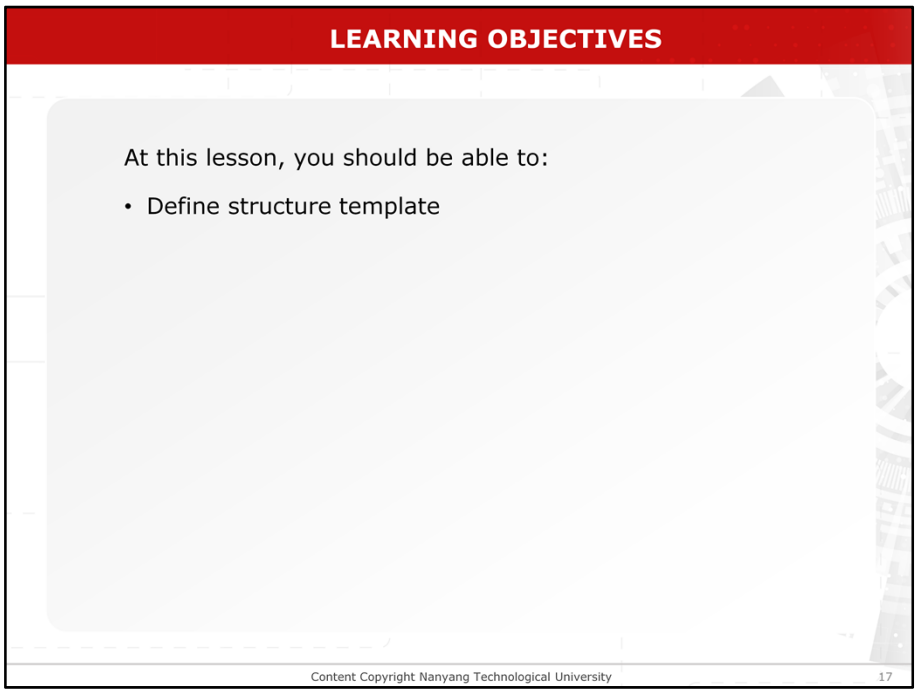


Structures

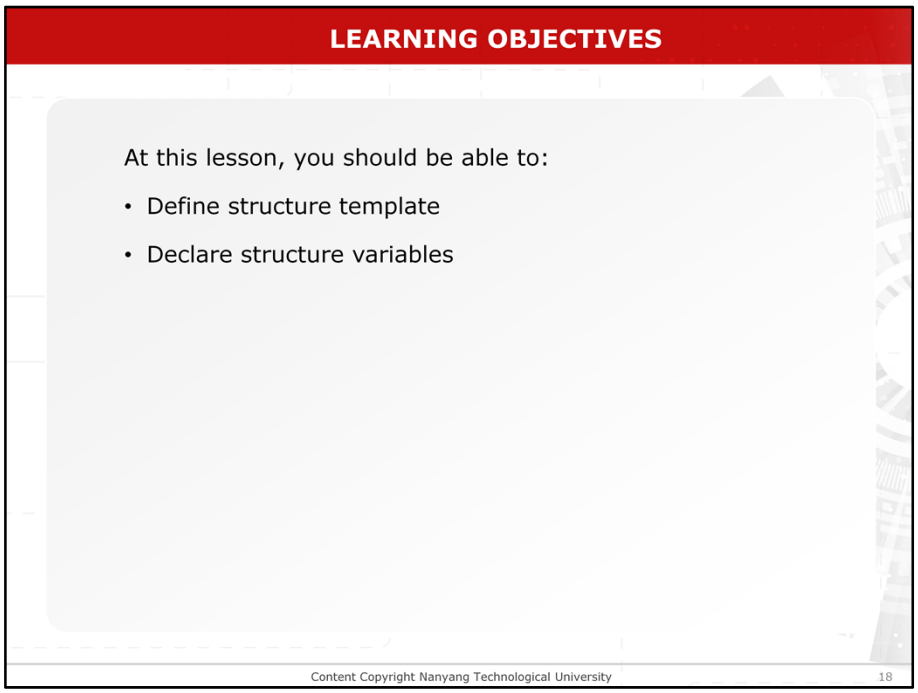
The following are the coverage for structures. This video is focused on the 1st topic: structure declaration, initialization and operations



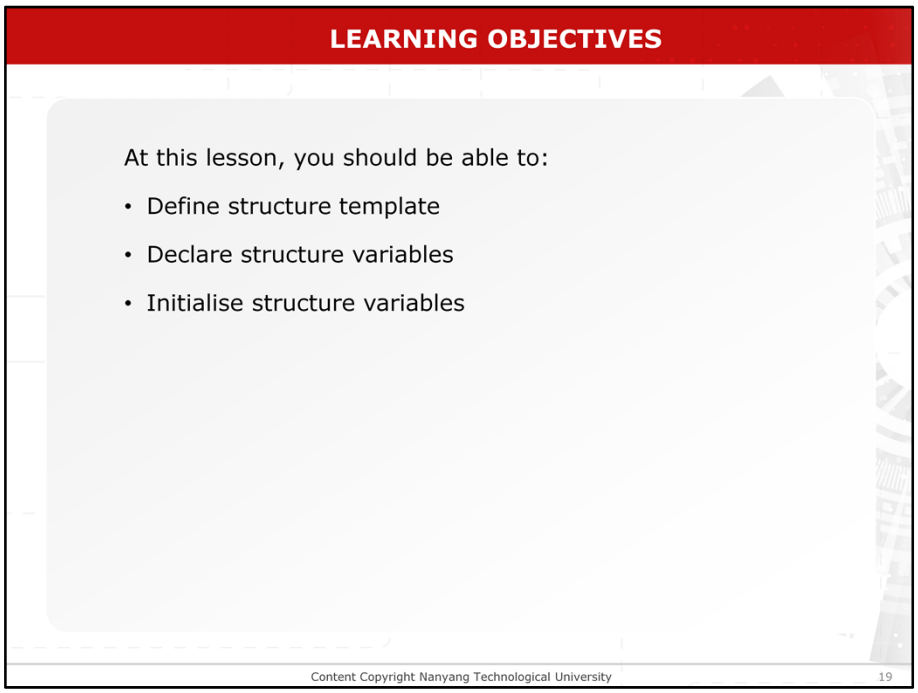
LEARNING OBJECTIVES: At this lesson, you should be able to:



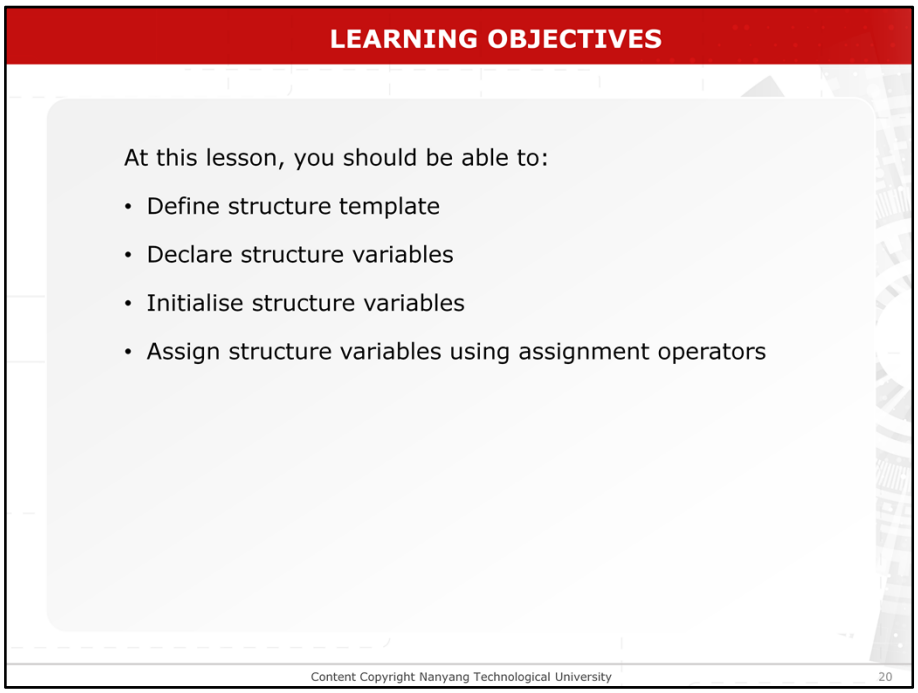
Define structure template



Declare structure variables



Initialise structure variables



Assign structure variables using assignment operators

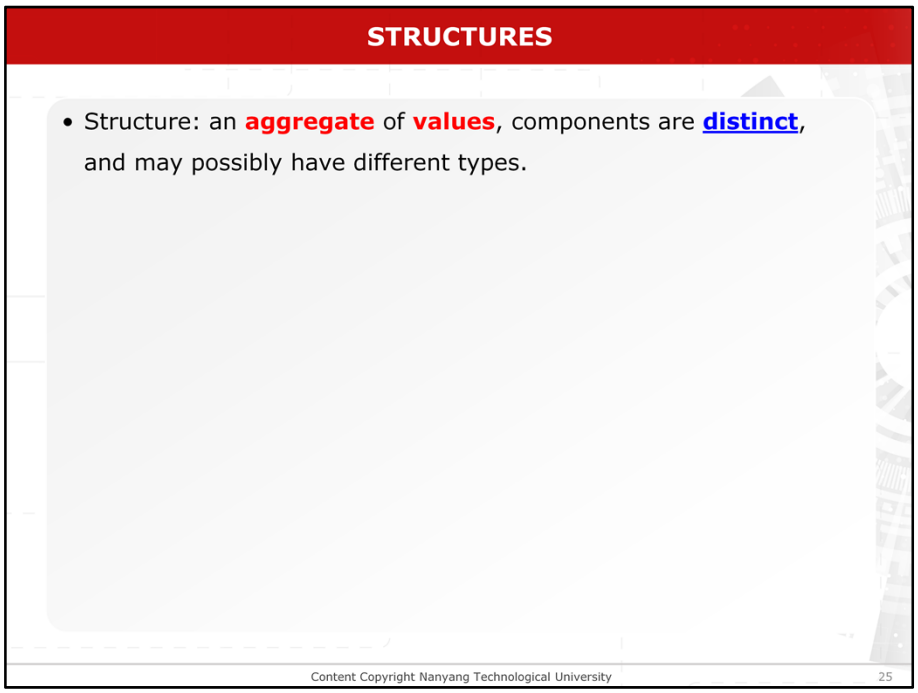
Records

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.



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:

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];

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];

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;

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]

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:

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);

Structures

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

STRUCTURES

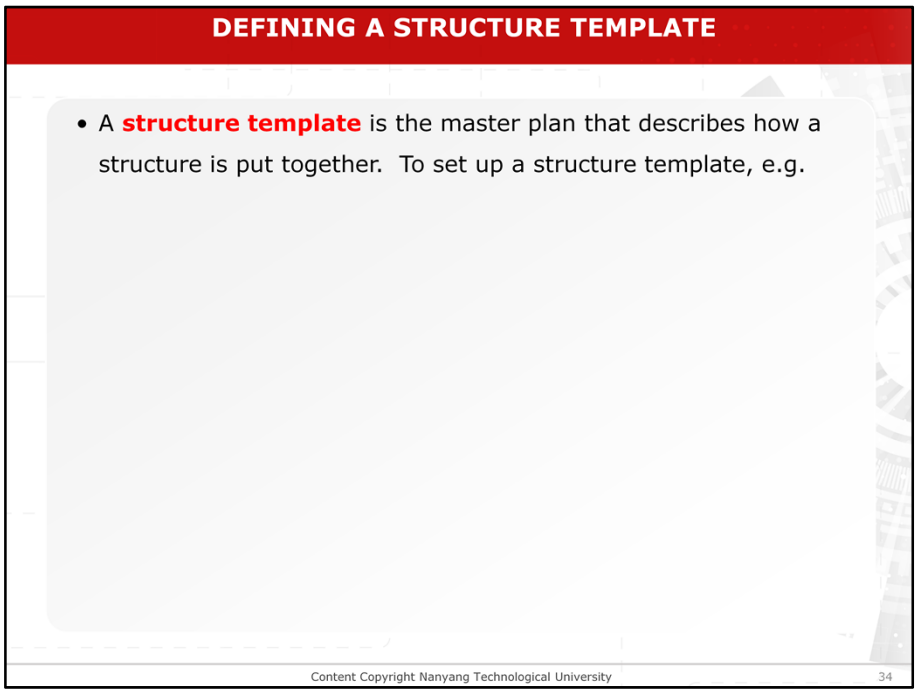
- Structure: an **aggregate** of **values**, components are **distinct**, and may possibly have different types.

- 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]

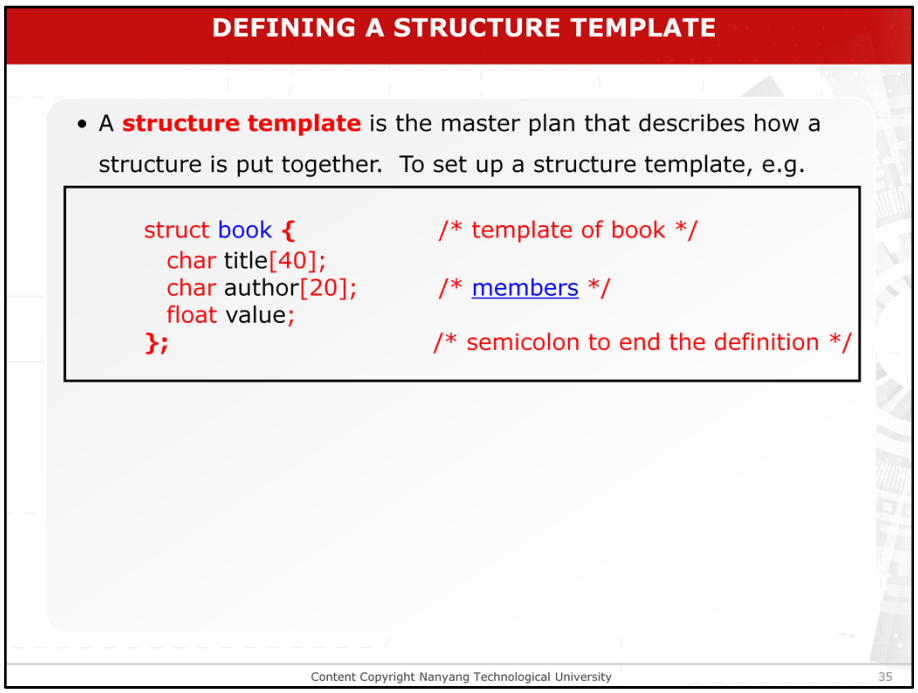
- Two steps to use a structure:
  - Setting up a structure template (similar to a data type);
  - Defining a variable on the structure template.

defining a variable based on the structure template



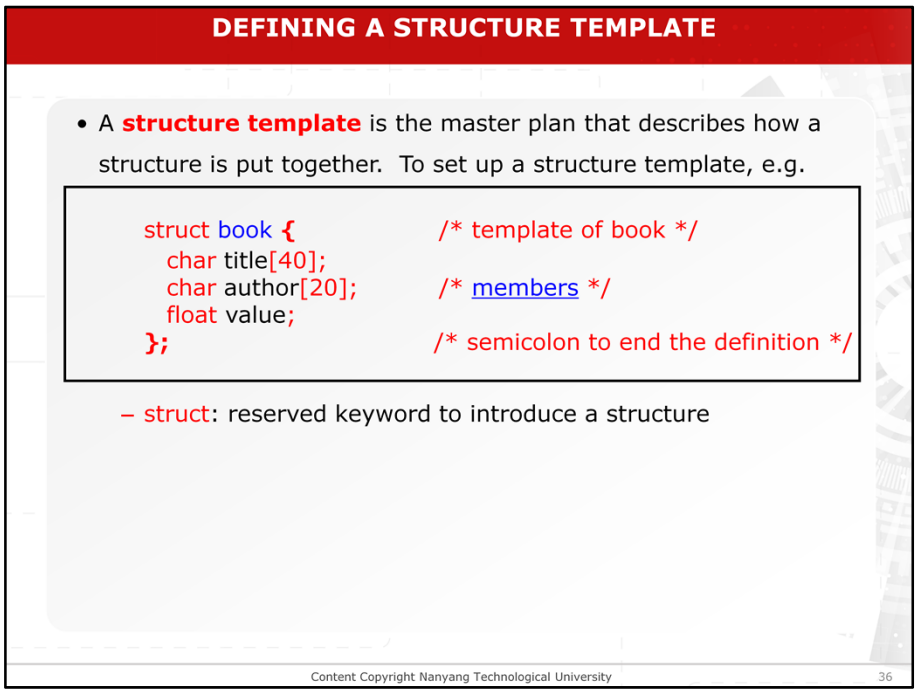
Structure Template (Data Type)

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:

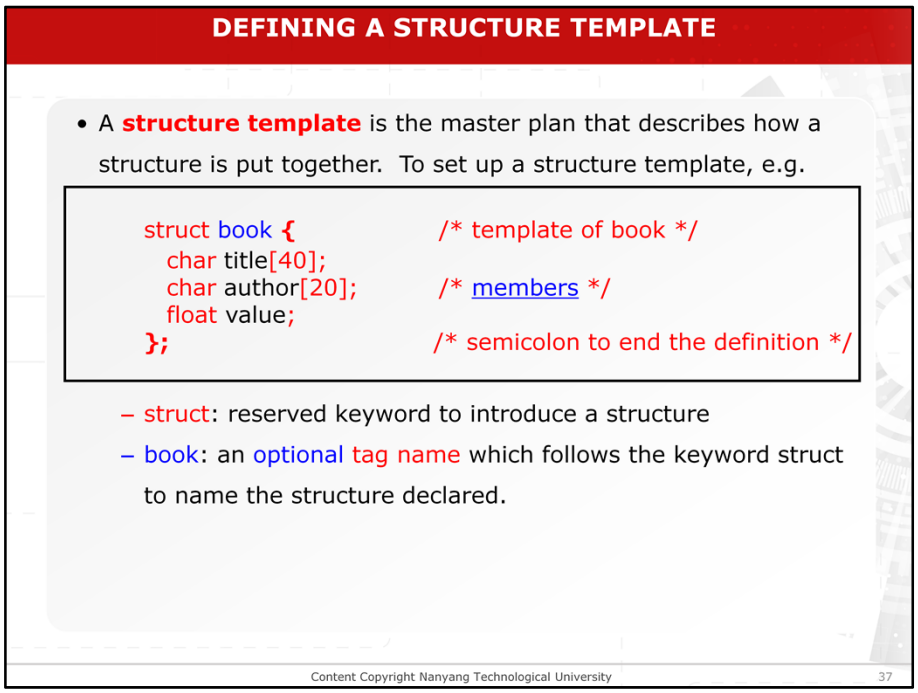


Structure Template (Data Type)

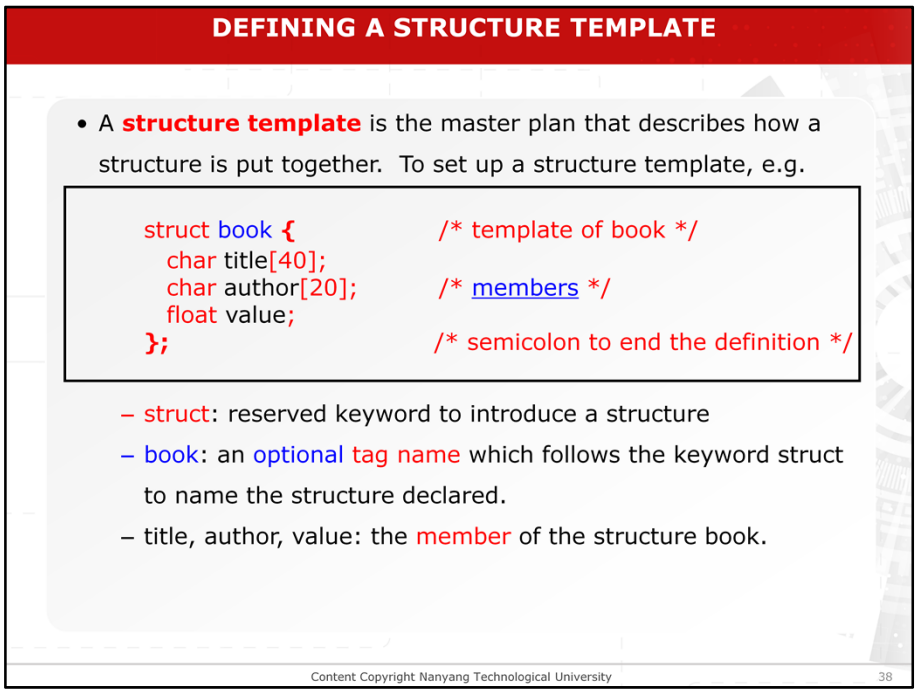
A structure template can be seen below



struct is a reserved keyword to introduce a structure.



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



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.

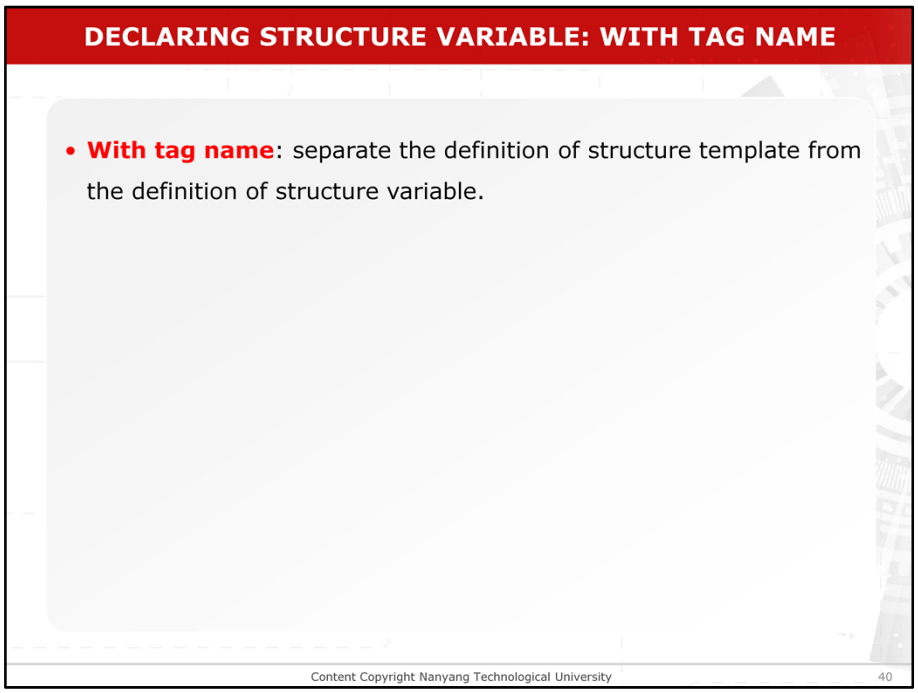
DEFINING A STRUCTURE TEMPLATE

- 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 */
```

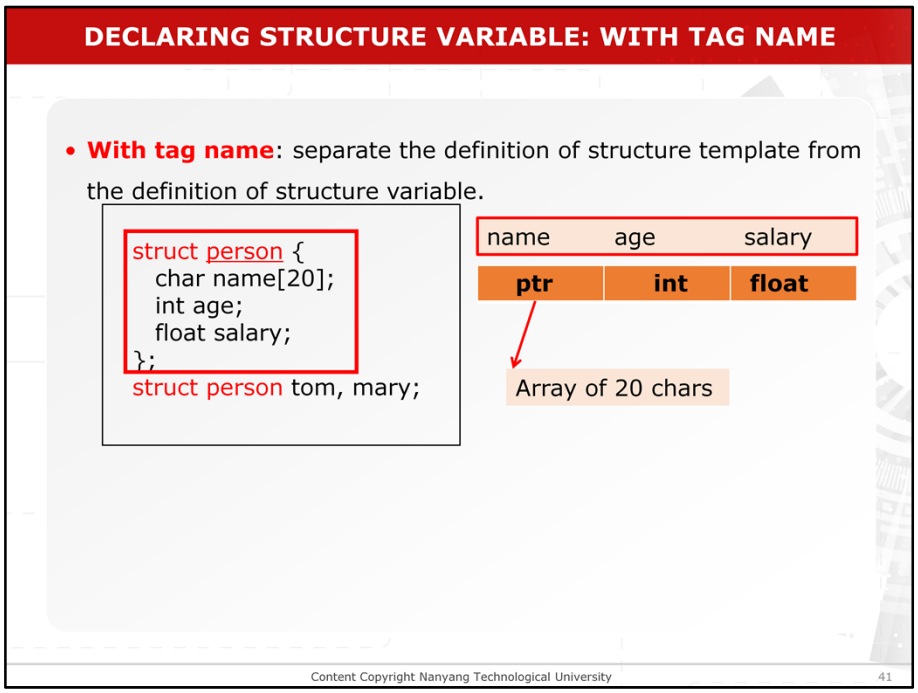
  - struct: reserved keyword to introduce a structure
  - book: an optional tag name which follows the keyword struct to name the structure declared.
  - title, author, value: the member of the structure book.
- Note - The above declaration just declares a template, not a variable. **No memory space**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

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.

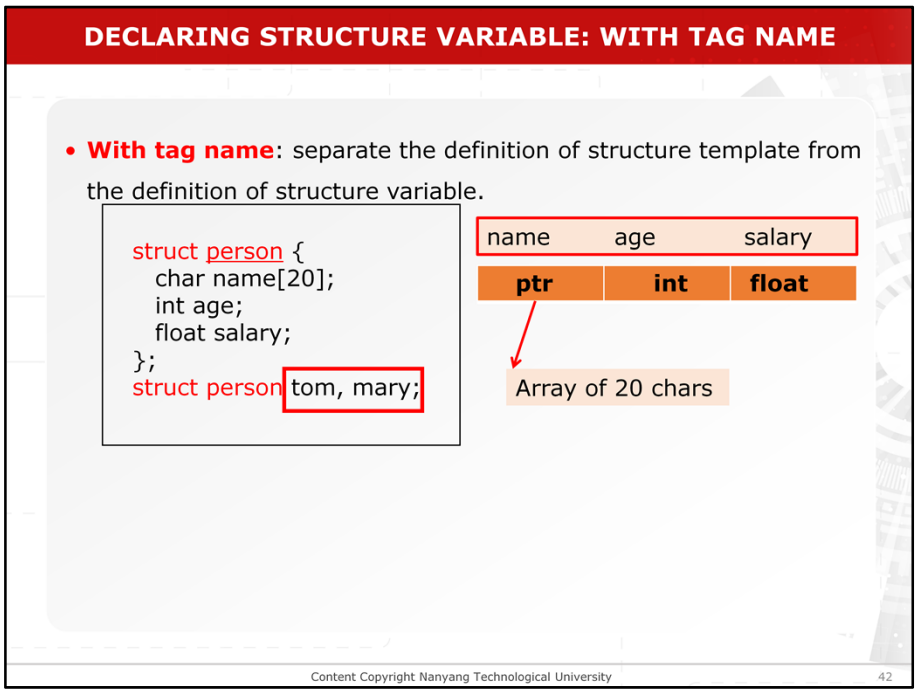


Structure Variable: with Tag Name

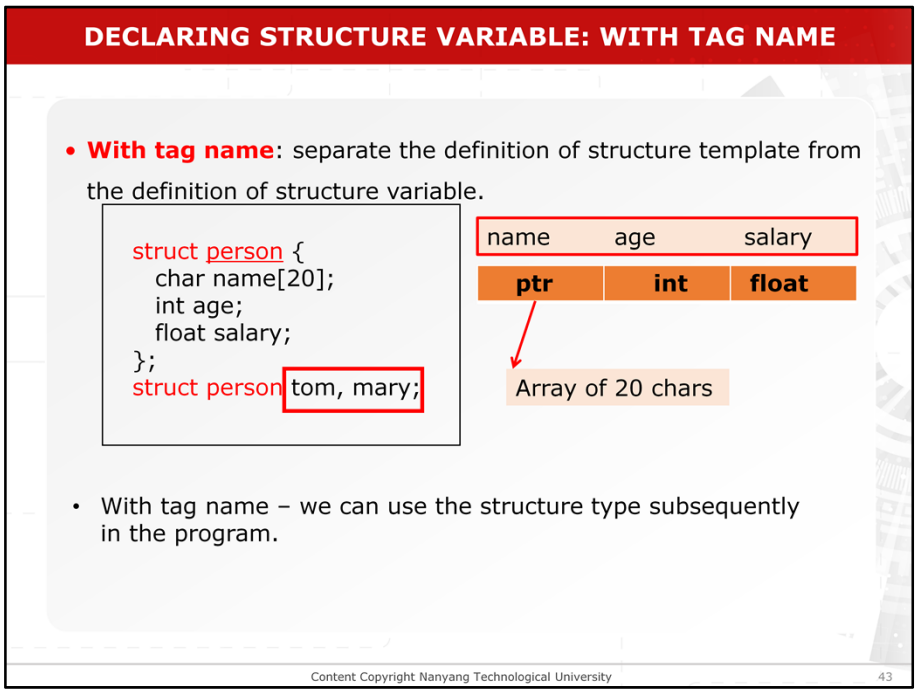
The structure name or tag is optional. With structure tag, the definition of structure template can be separated from the definition of structure variables.



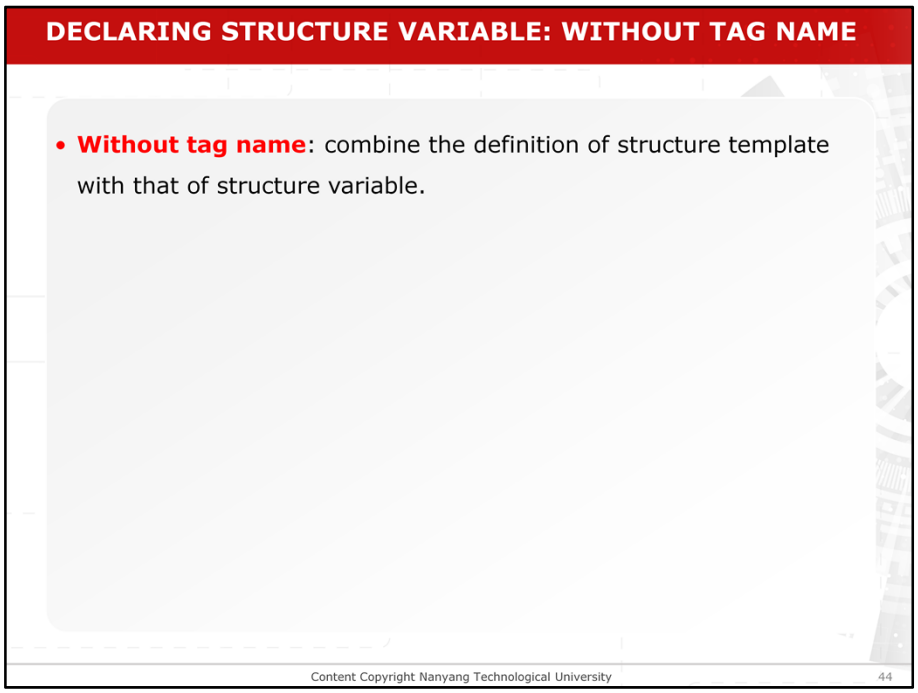
In the declaration shown here, a structure template **person** comprising three components is created.



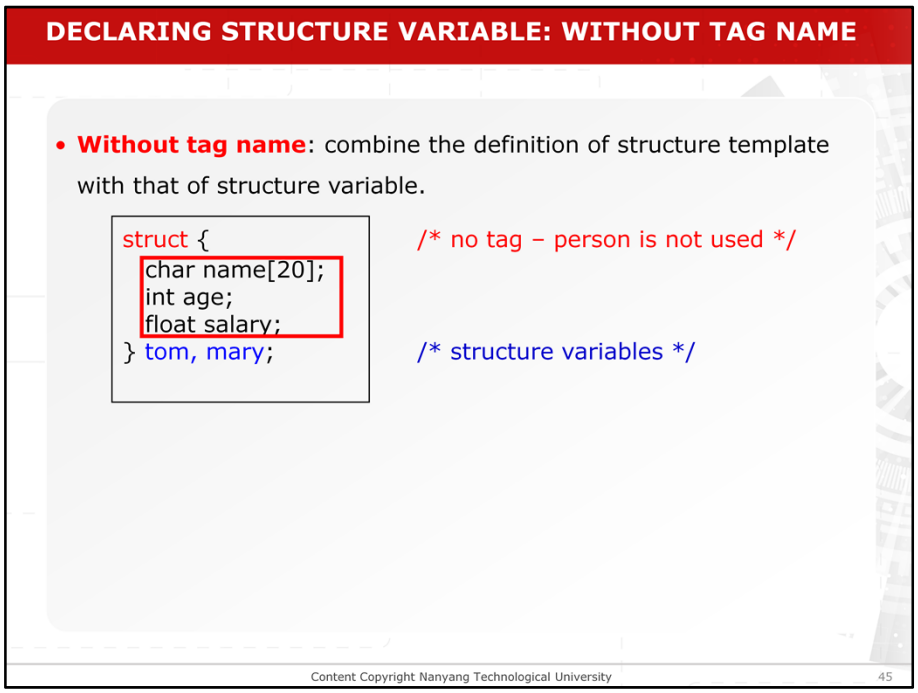
tom and **mary** are two structure variables which are declared using the structure **person**. With tag name, we can use the structure data type subsequently in the program.



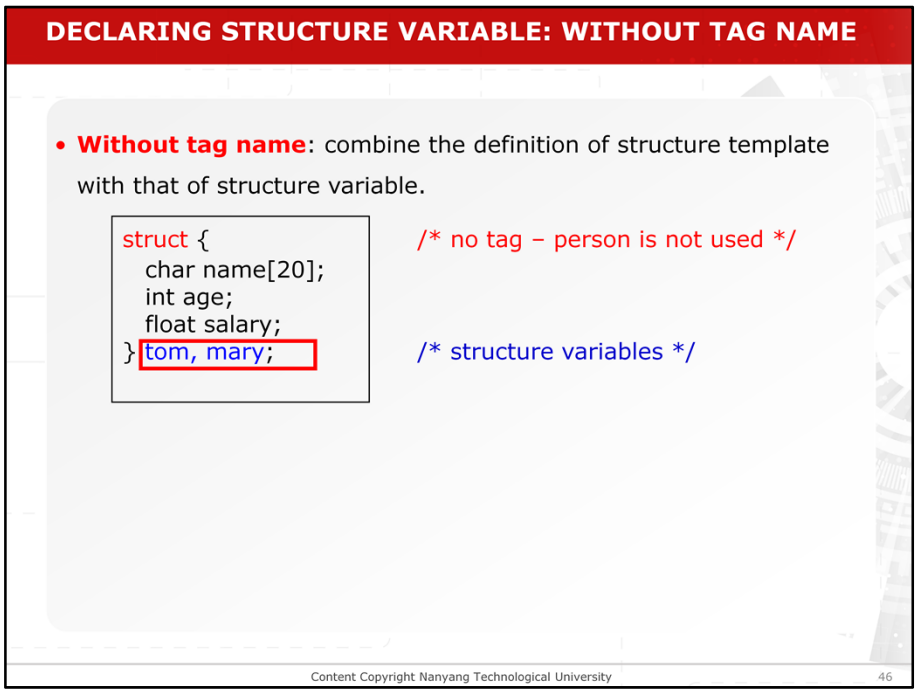
With tag name, we can use the structure data type subsequently in the program.



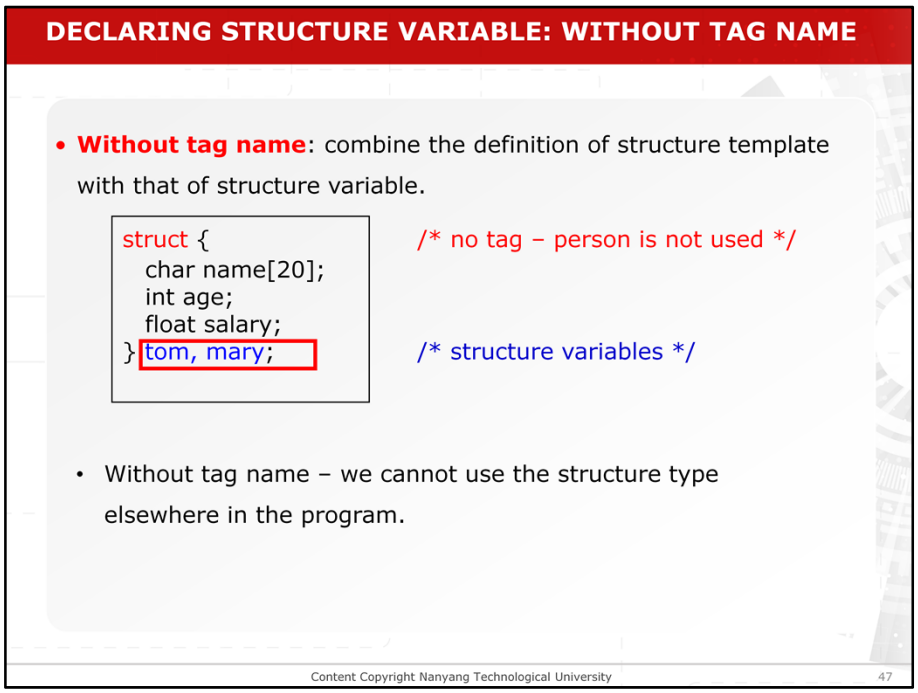
Declaring Structure Variable without Tag Name
Without structure tag, the definition of structure template must be combined with that of structure variables.



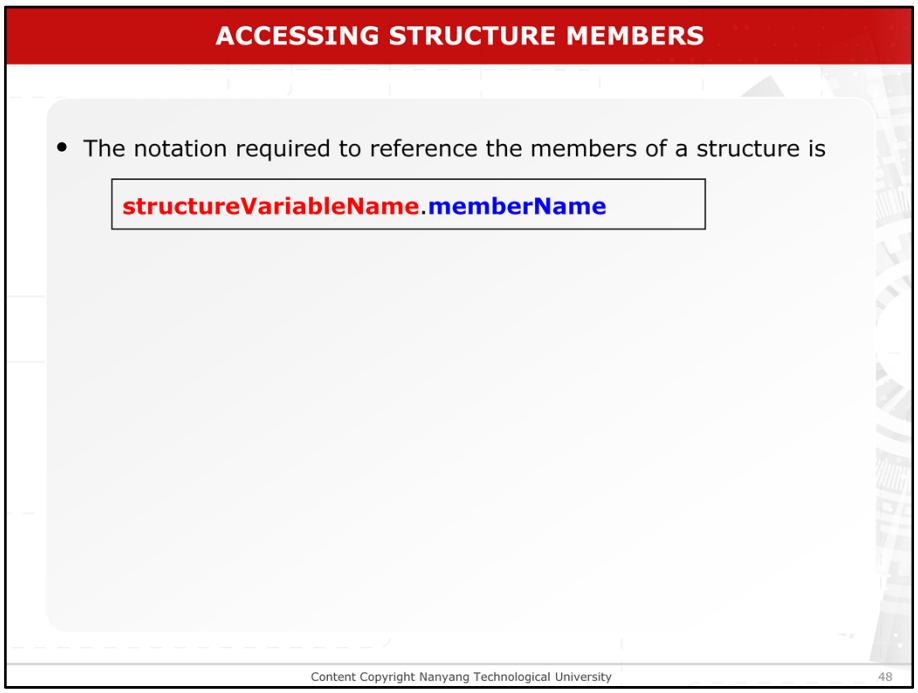
In the declaration shown, a structure template is created with the three components:



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.

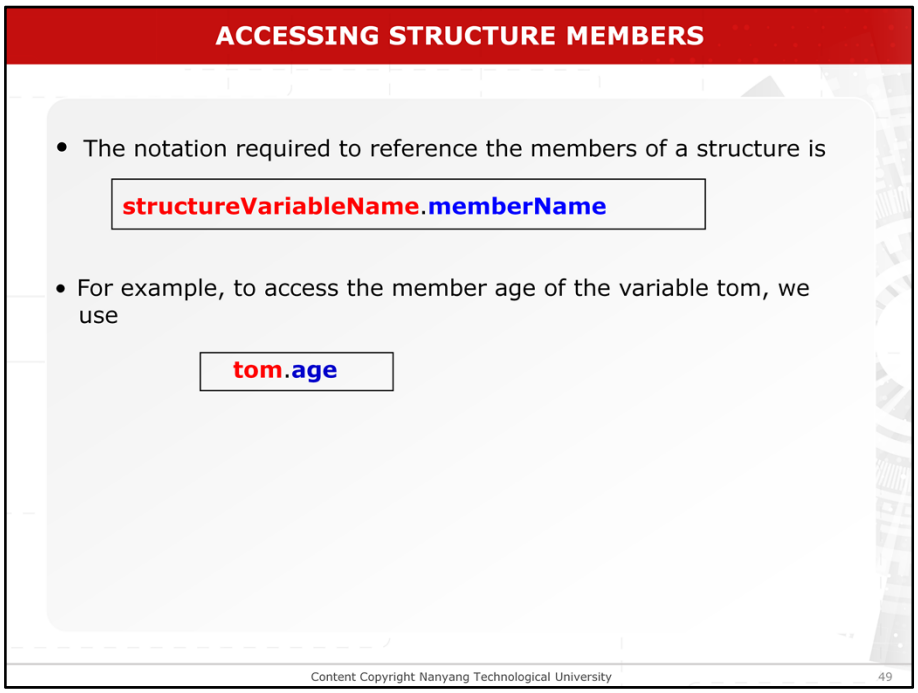


Without structure tag name, we cannot use the structure elsewhere in the program. It is always a good idea to include a structure tag when defining a structure.

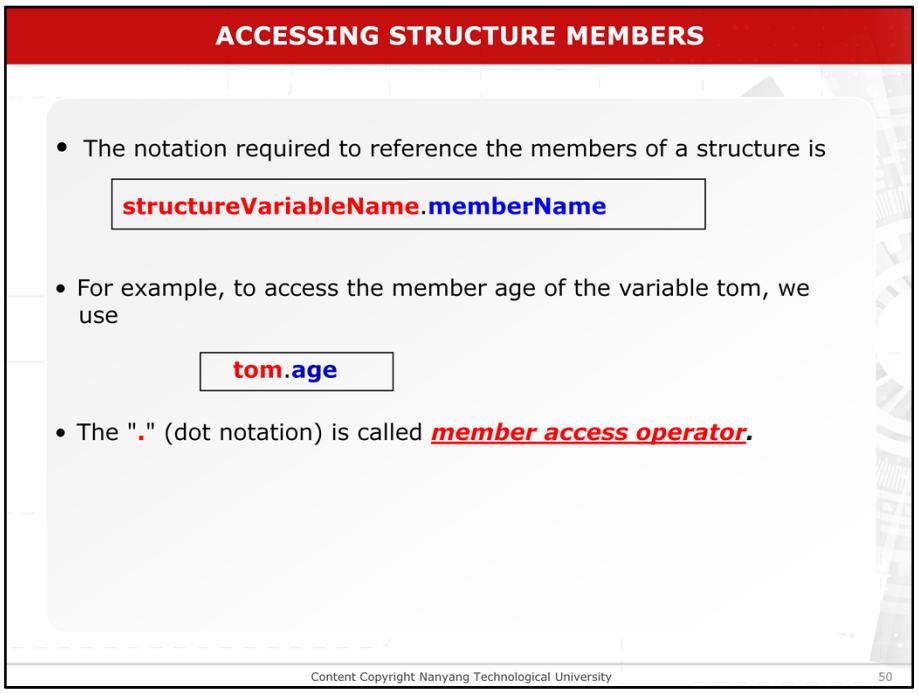


Accessing Structure Members

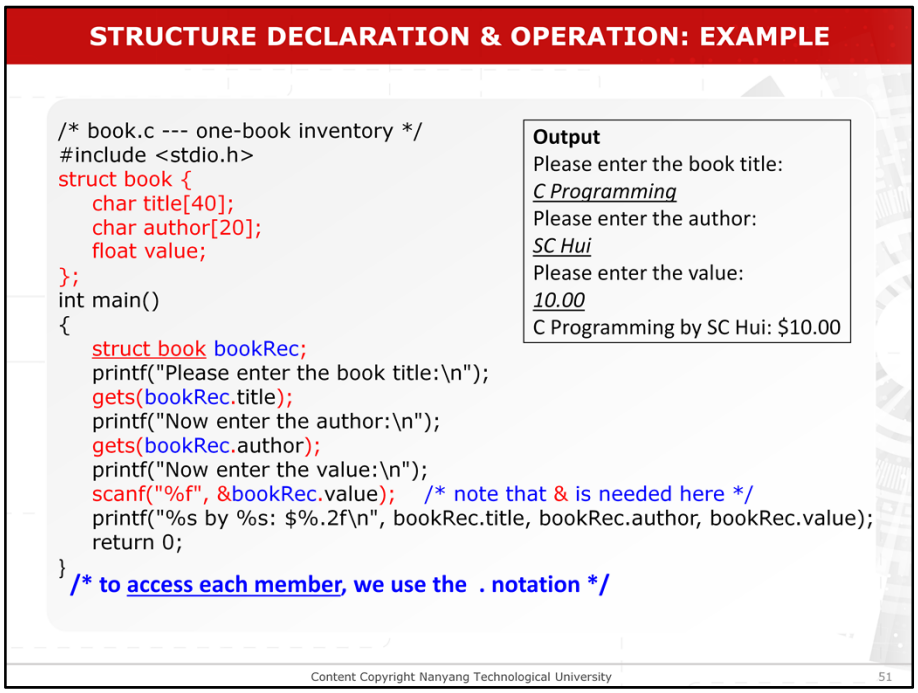
The notation required to access a member of a structure is shown here



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



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



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;
}
```


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;
}
```


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;
}
```


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;
}
```


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;
}
```


The scan f 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

 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
```


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
```


and book value will then be printed on the screen.

STRUCTURE DECLARATION & OPERATION: EXAMPLE

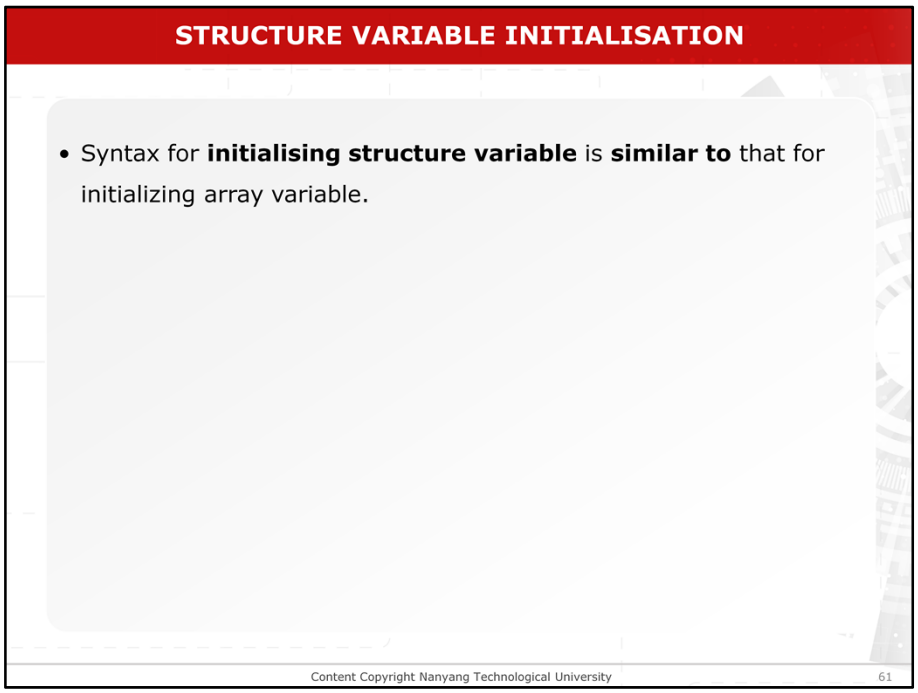
```
/* book.c --- one-book inventory */
#include <stdio.h>
struct book {
    char title[40];
    char author[20];
    float value;
};
int main()
{
    struct book bookRec;
    printf("Please enter the book title:\n");
    gets(bookRec.title);
    printf("Now enter the author:\n");
    gets(bookRec.author);
    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
```

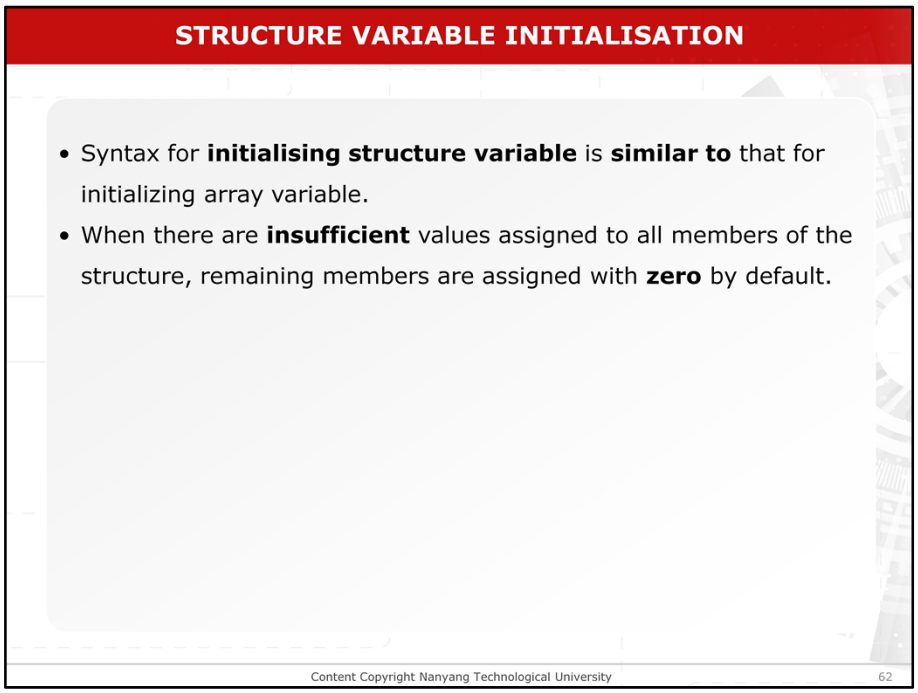
Q: What is the memory size of the variable bookRec?

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.

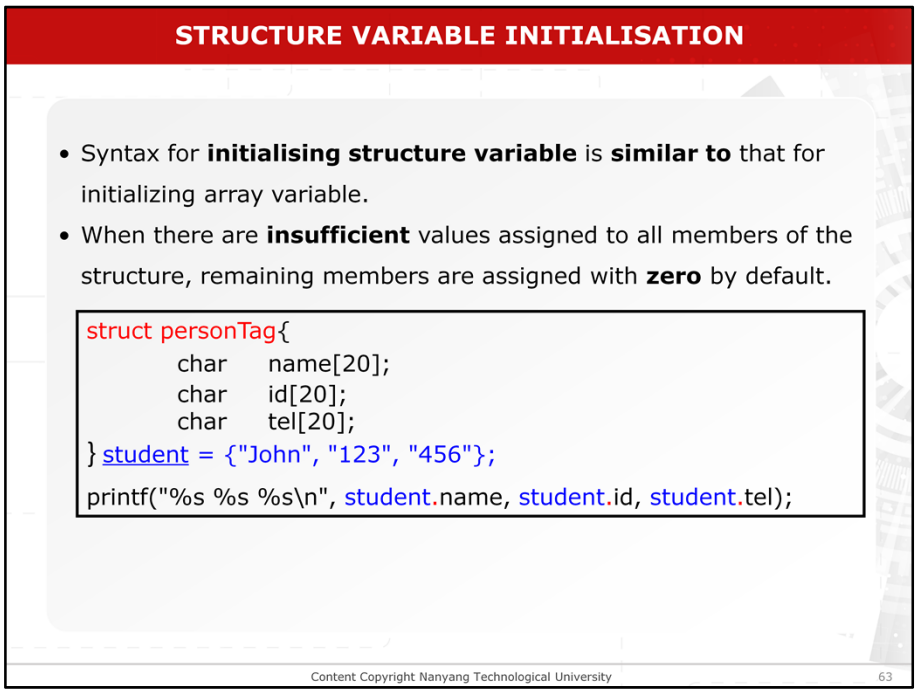


Structure Variable Initialization and Operation

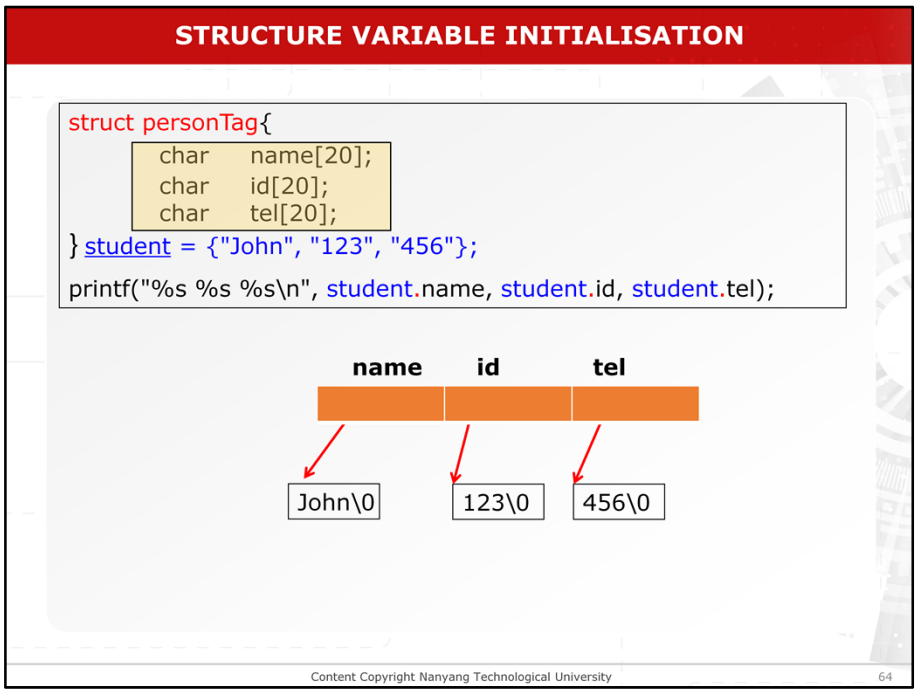
The syntax for initializing structure variable is similar to that of initializing array variable



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



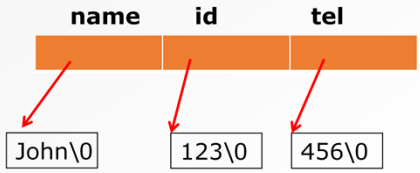
. The structure variable is followed by an assignment symbol and a list of values defined within braces:



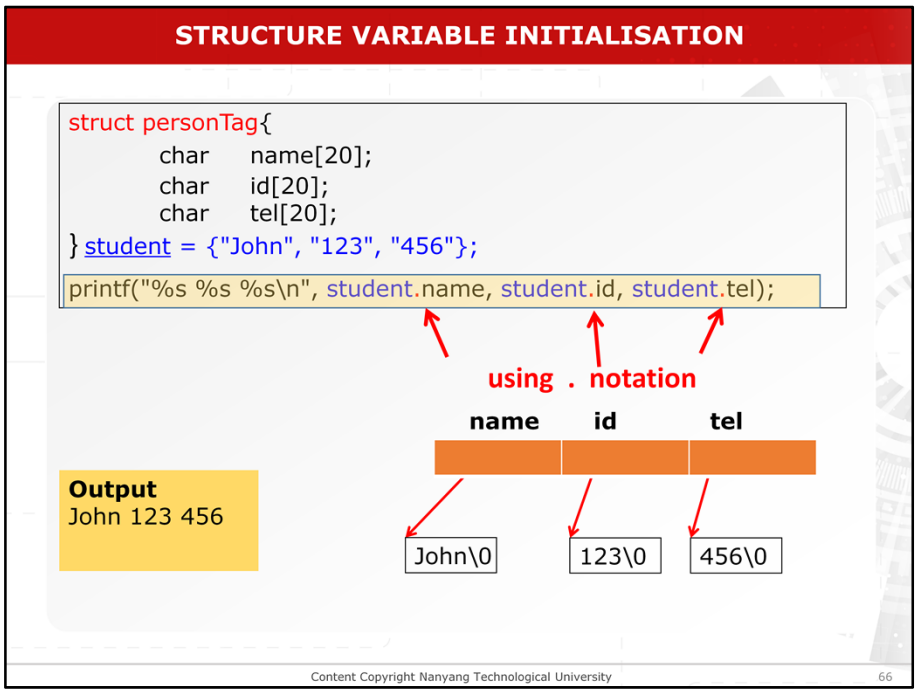
Initialization of variables can only be performed with constant values or constant expressions that deliver a value of the required type. The initial values are assigned to the individual members of the structure in the order in which the members occur.

STRUCTURE VARIABLE INITIALISATION

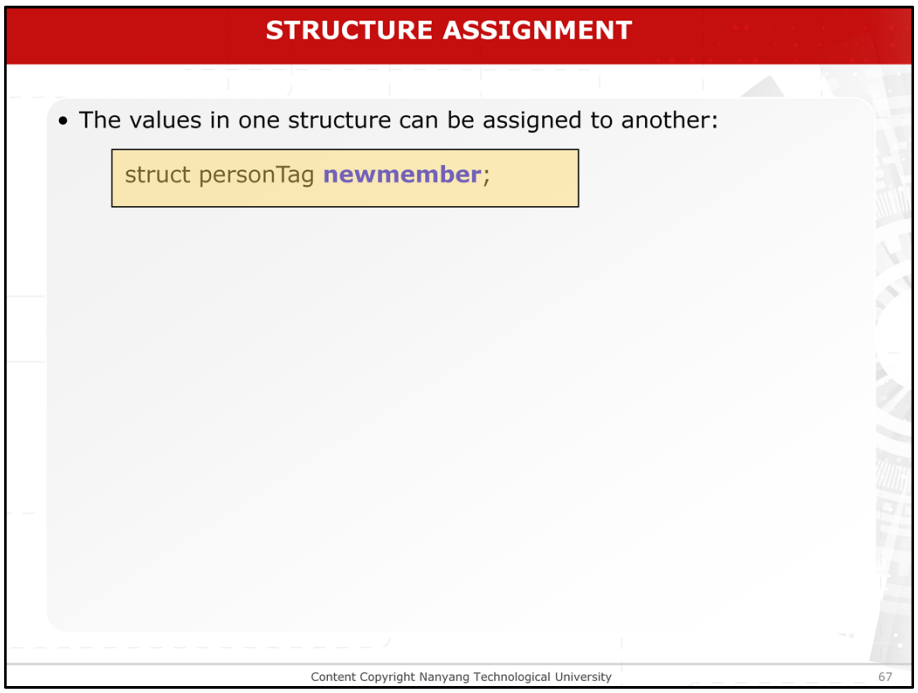
```
struct personTag{
        char    name[20];
        char    id[20];
        char    tel[20];
} student = {"John", "123", "456"};

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



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

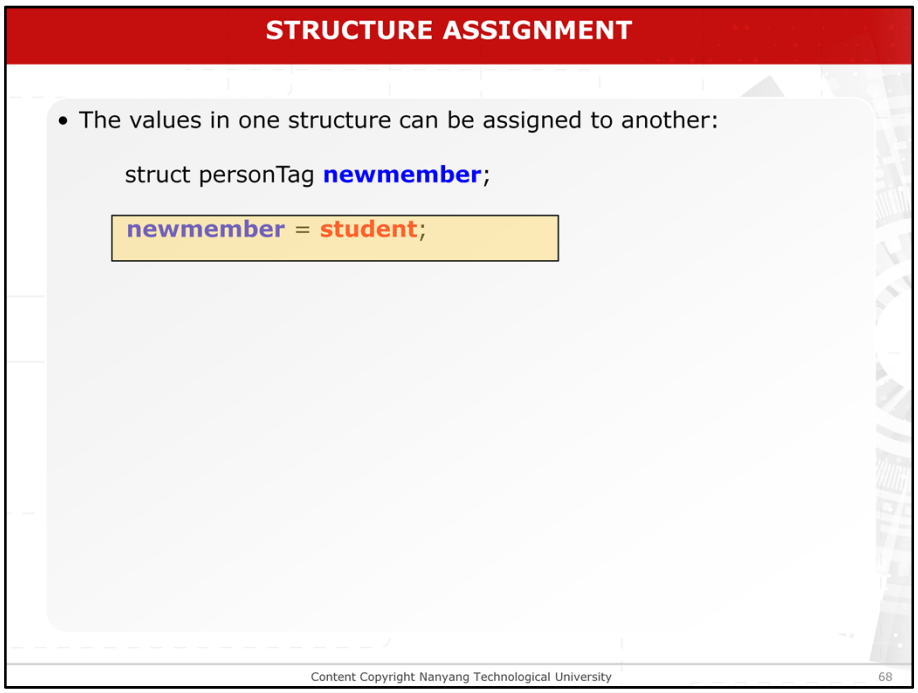


To print the data of the structure variable **student**, the highlighted statement and the dot notation is used in order to access the member of the structure.

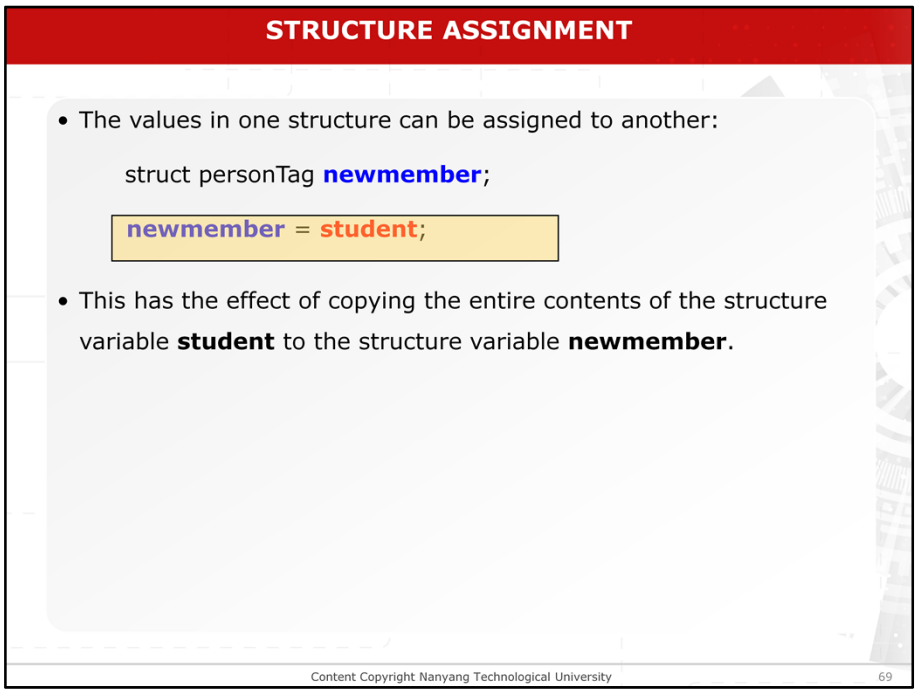


Structure Assignment

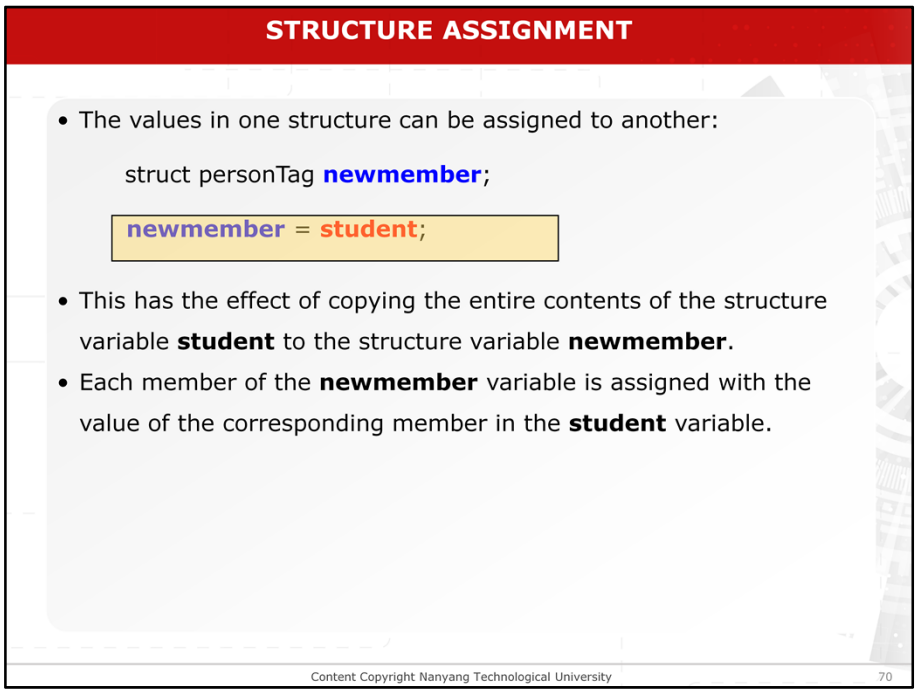
The value of one structure variable can be assigned to another structure variable of the same type using the assignment operator. First, we define a new variable **newmember** under the data type **struct personTag**:



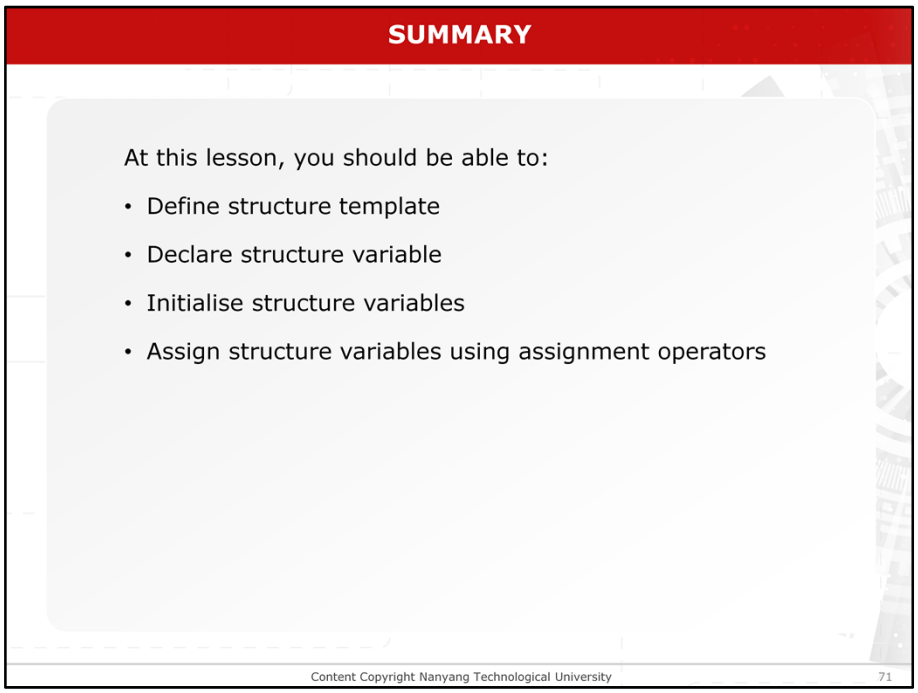
Then, we can assign the **struct personTag** variable **student** to **newmember** as shown;



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.



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