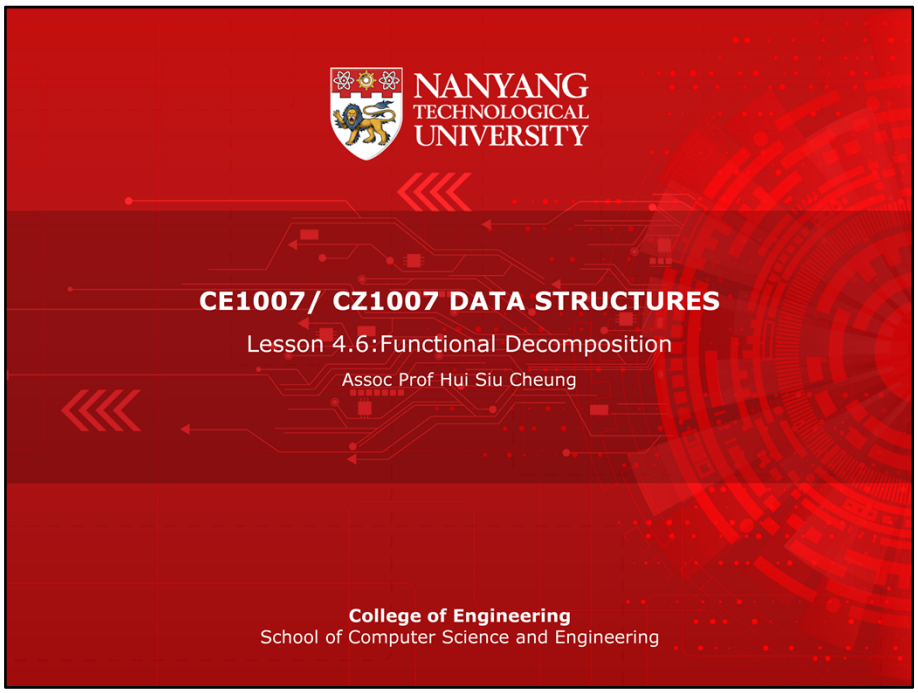
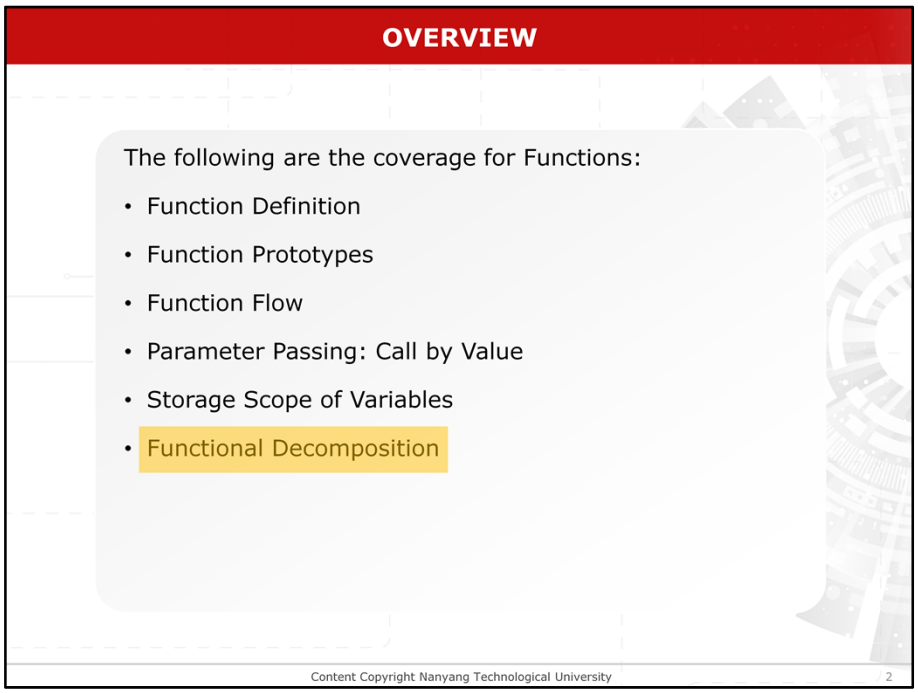
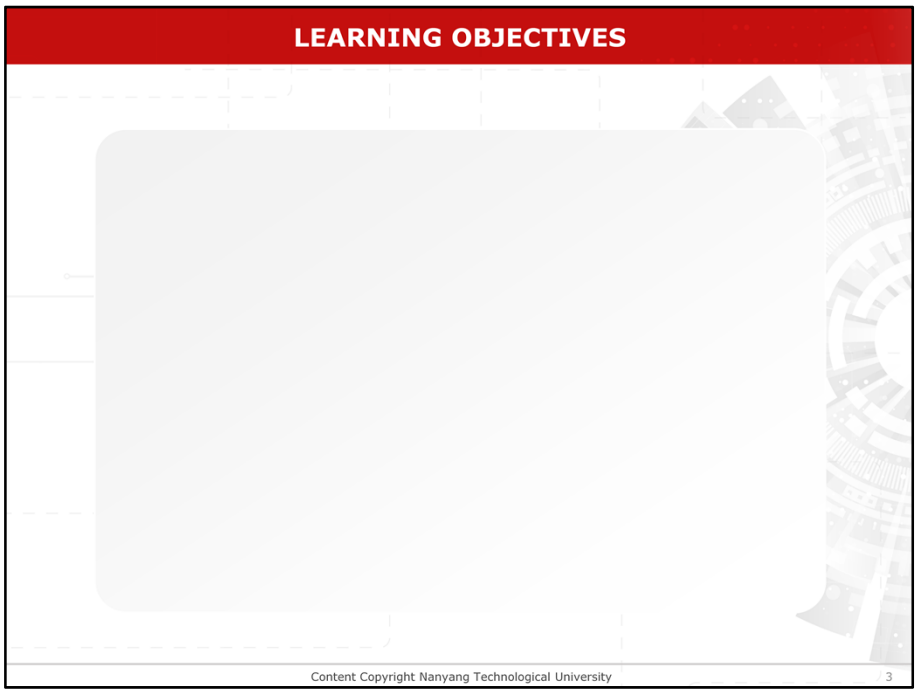


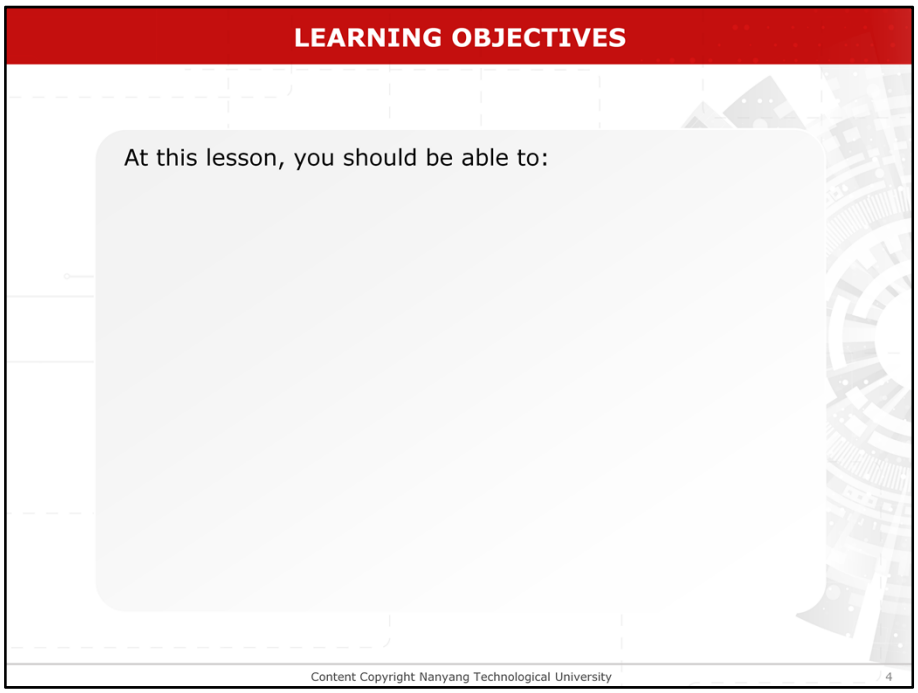
This lesson is on functions



There are 6 main sections to cover for Functions. This video focused on the 5th topic: storage scope of variables



Learning objectives



At this lesson, you should be able to:

LEARNING OBJECTIVES

At this lesson, you should be able to:

- Explain the concept of function decomposition

 5

Explain the concept of function decomposition

LEARNING OBJECTIVES

At this lesson, you should be able to:

- Explain the concept of functional decomposition
- Write program using functions

Write program using functions

FUNCTIONAL DECOMPOSITION

- Functional decomposition basically means the top-down stepwise refinement technique that uses the divide-and-conquer strategy.

Functional decomposition basically means the top-down stepwise refinement technique that uses the divide-and-conquer strategy.

FUNCTIONAL DECOMPOSITION

- Functional decomposition basically means the top-down stepwise refinement technique that uses the divide-and-conquer strategy.
- It starts with the high level description of the program and decomposes the program into successively smaller components until we arrive at a set of suitably sized functions (or algorithms).

It starts with the high level description of the program and decomposes the program into successively smaller components until we arrive at a set of suitably sized functions (or algorithms).

FUNCTIONAL DECOMPOSITION

- Functional decomposition basically means the top-down stepwise refinement technique that uses the divide-and-conquer strategy.
- It starts with the high level description of the program and decomposes the program into successively smaller components until we arrive at a set of suitably sized functions (or algorithms).
- Then, we design the code for the individual functions using stepwise refinement.

 9

Then, we design the code for the individual functions using stepwise refinement.

FUNCTIONAL DECOMPOSITION

- At each level of refinement, we are only concerned with what the lower level functions will do.

 10

At each level of refinement, we are only concerned with what the lower level functions will do.

FUNCTIONAL DECOMPOSITION

- At each level of refinement, we are only concerned with what the lower level functions will do.
- Functional decomposition produces smaller functions that are easier to understand.

 11

Functional decomposition produces smaller functions that are easier to understand.

FUNCTIONAL DECOMPOSITION

- At each level of refinement, we are only concerned with what the lower level functions will do.
- Functional decomposition produces smaller functions that are easier to understand.
- Smaller functions promote software reusability.

 12

Smaller functions promote software reusability.

FUNCTIONAL DECOMPOSITION

- At each level of refinement, we are only concerned with what the lower level functions will do.
- Functional decomposition produces smaller functions that are easier to understand.
- Smaller functions promote software reusability.
- In general, functions should be small, so that they can be developed and tested separately.

 13

In general, functions should be small, so that they can be developed and tested separately.

FUNCTIONAL DECOMPOSITION

- At each level of refinement, we are only concerned with what the lower level functions will do.
- Functional decomposition produces smaller functions that are easier to understand.
- Smaller functions promote software reusability.
- In general, functions should be small, so that they can be developed and tested separately.
- They should also be independent of each other.

 14

They should also be independent of each other.

FUNCTIONAL DECOMPOSITION: EXAMPLE

```
#include  <stdio.h>
#define  ...

int main()
{
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
} /* end.  line 2000 */
```

```
#include  <stdio.h>
#define  ...

int main()
{
.....
} /* line 20 */

float f1(float h)
{
.....
} /* line 55*/
......
void f18()
{
......
} /* line 1560 */
```

 15

In the program, it is decomposed into a number of smaller functions.

FUNCTIONAL DECOMPOSITION: EXAMPLE

```
#include <stdio.h>
#define ...

int main()
{
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
} /* end. line 2000 */
```

```
#include <stdio.h>
#define ...

int main()
{
.....
} /* line 20 */

float f1(float h)
{
.....
} /* line 55*/
......
void f18()
{
.....
} /* line 1560 */
```

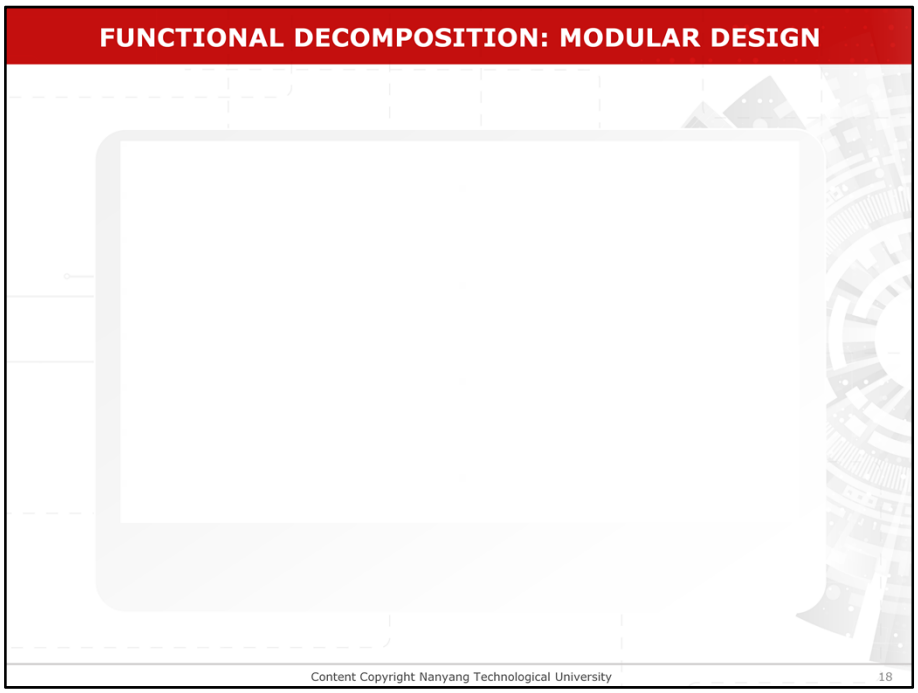
 16

The **main()** function will start the execution of the program and call other functions to perform different required operations.

FUNCTIONAL DECOMPOSITION: EXAMPLE

```
#include  <stdio.h>
#define  ...

int main()
{
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
} /* end.  line 2000 */
```

```
#include  <stdio.h>
#define  ...

int main()
{
.....
} /* line 20 */

float f1(float h)
{
.....
} /* line 55*/
.....

void f18()
{
.....
} /* line 1560 */
```

Functional Decomposition: Modular Design

Using the functional decomposition and top-down stepwise refinement technique, a problem is broken up into a number of smaller subproblems or functions. We then develop the algorithms for the functions. These functions can then be implemented using a programming language such as C. These functions are also called *modules*. This approach of designing programs as functional modules is called *modular design*. The functions or modules should be small and self-contained, so that they can be developed and tested separately. They should also be independent of each other. There are a number of advantages for modular design. Modular programs are easier to write and debug, since they can be developed and tested separately. Another advantage is that modular programs can be developed by different programmers as each programmer can work on a single module of the program independently. Moreover, a library of modules can be developed which can then be reused in other programs that require the same implementation. This can reduce program development time and enhances program reliability. Therefore, modular design can simplify program development significantly.

In the figure, it shows a typical structure of a program consisting of the main function and other functions for solving a problem. Usually, the functions could be quite complex as well, and they can be divided further into smaller functions.

PROBLEM: WRITING A C PROGRAM WITH A FUNCTION

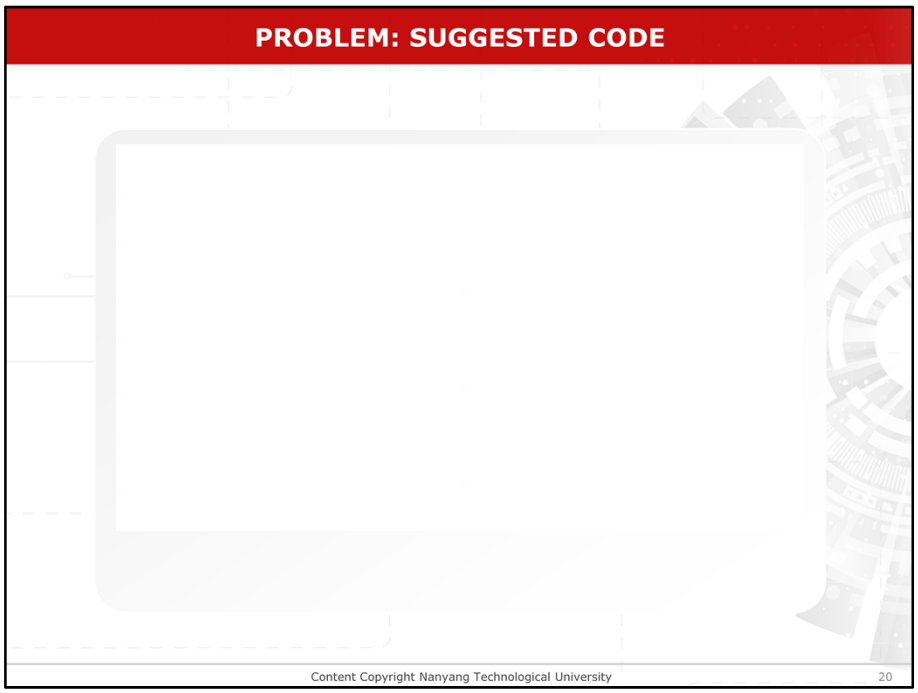
```
/* Purpose: Write a function to find the maximum of two numbers.
The main() function calls the findMax() function using call by value */
#include <stdio.h>
float findMax(float x, float y);
int main()
{
  float x,y, max;
  printf("Enter 2 numbers: ");
  scanf("%f %f", &x, &y);
  max = findMax(x, y);
  printf("The max is %f.\n", max);
  return 0;
}

float findMax(float x, float y)
{
  float maxnum;
  /* write your code here */
}
```

OUTPUT

 19

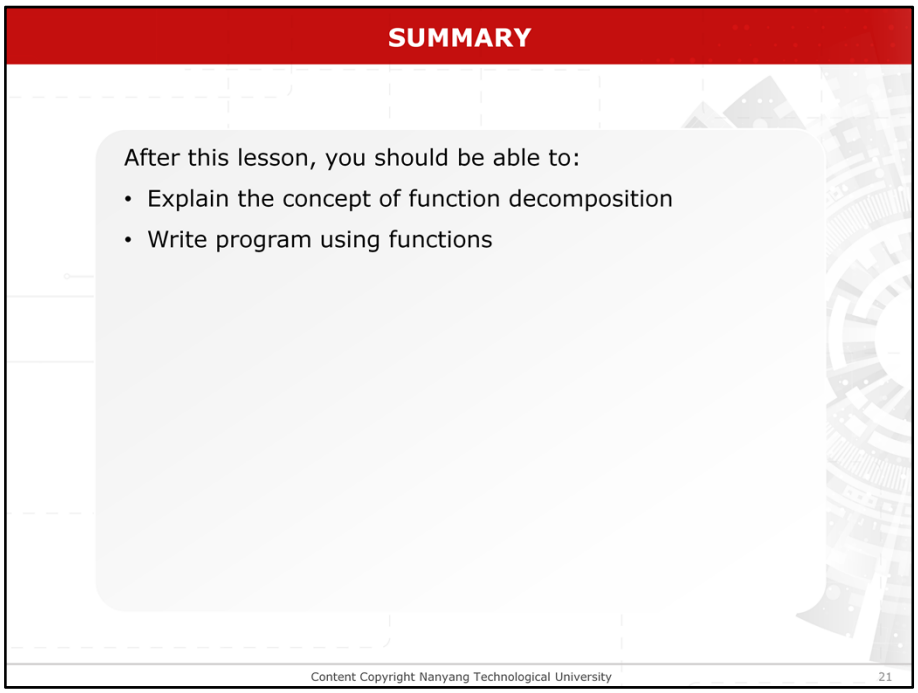
<u>Programming Problem</u>

The purpose of the program is to compute the maximum of two floating point numbers. Write a function called **findMax()** to achieve this purpose. The function will take in two arguments in floating point type, compute the maximum of the two values and return the maximum number to the calling function.



Programming Problem: Suggested Code

The suggested code for the function findMax() is given. It will take in two arguments **x** and **y** from the calling function, compute the maximum number and returns the maximum number to the calling function. The calling function **main()** will then stores the returned result in the variable **max** which is then printed on the screen.



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