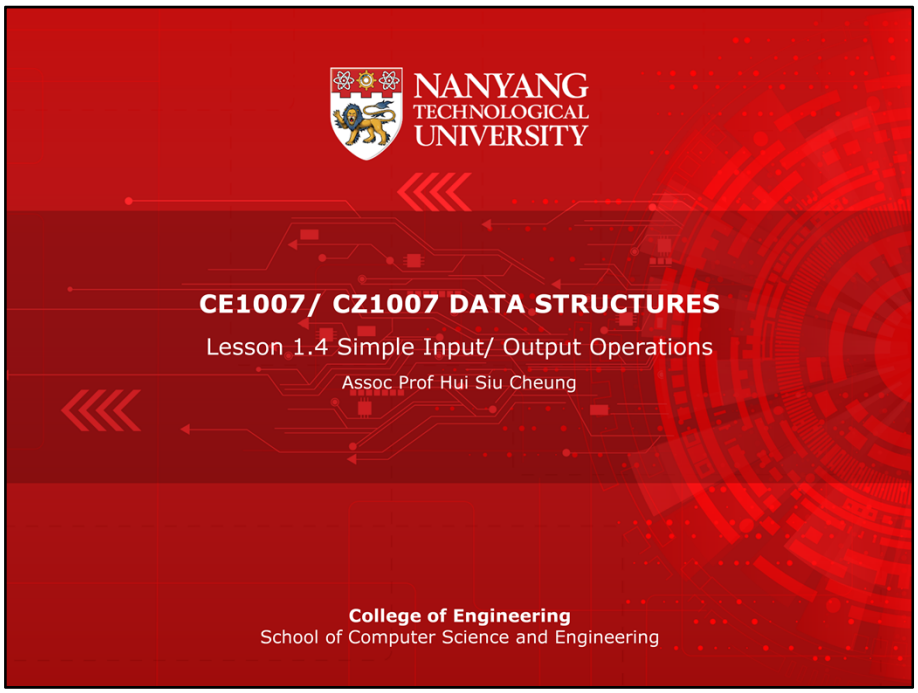
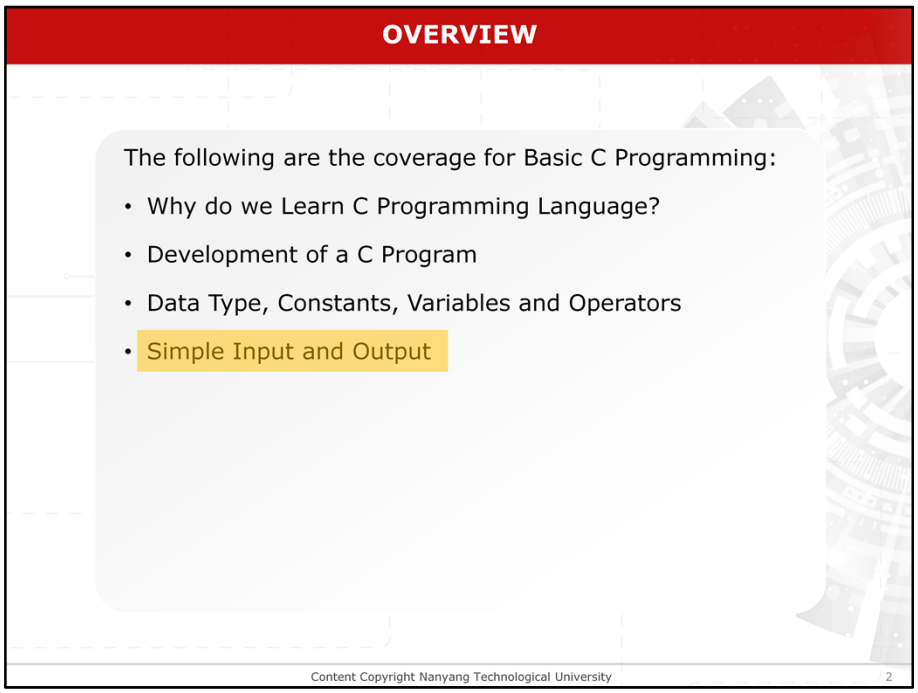
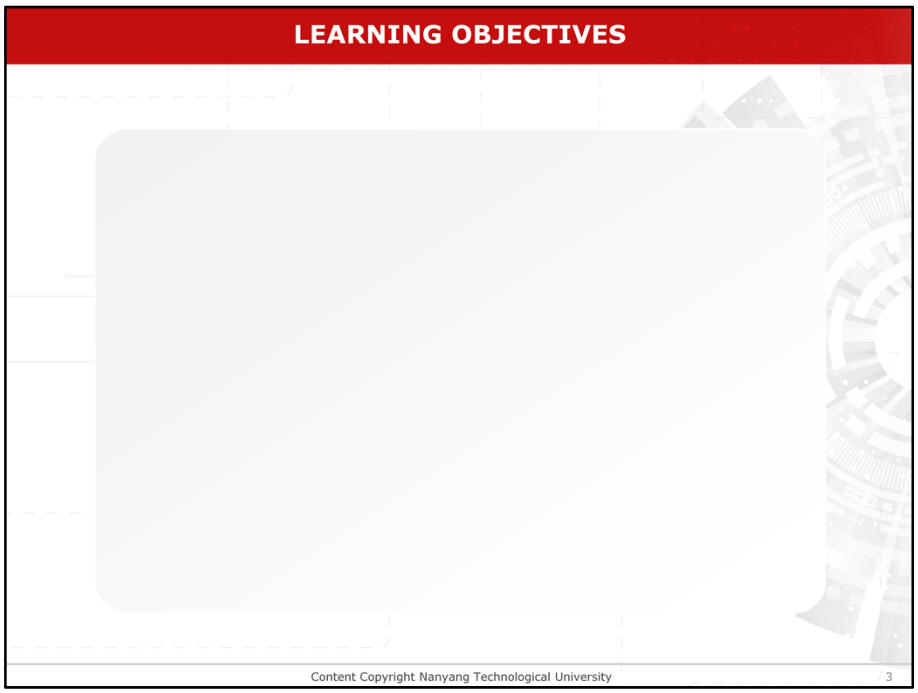


This lesson is on Basic C-Programming

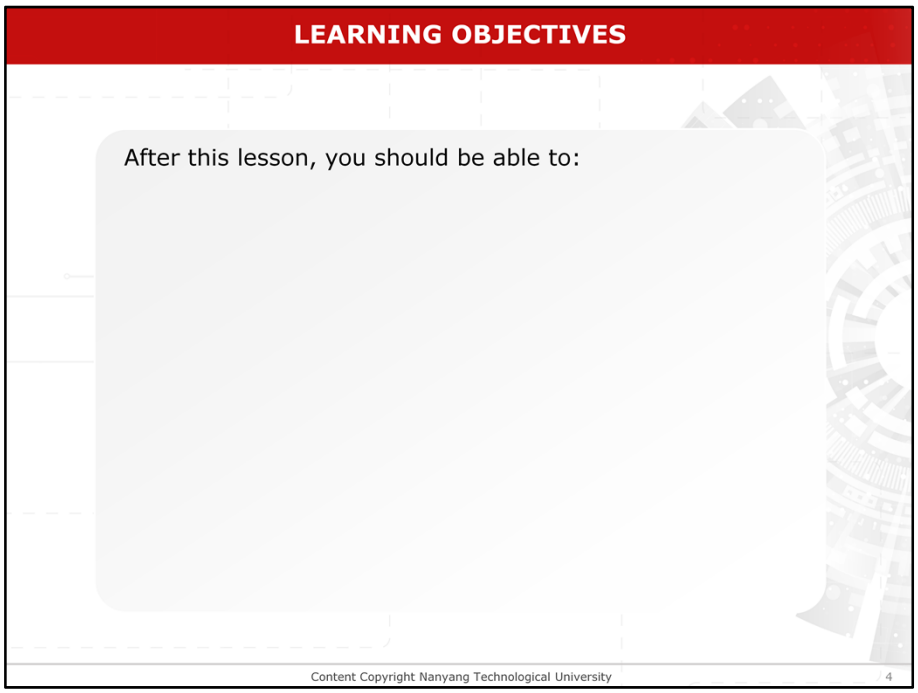


Basic C Programming

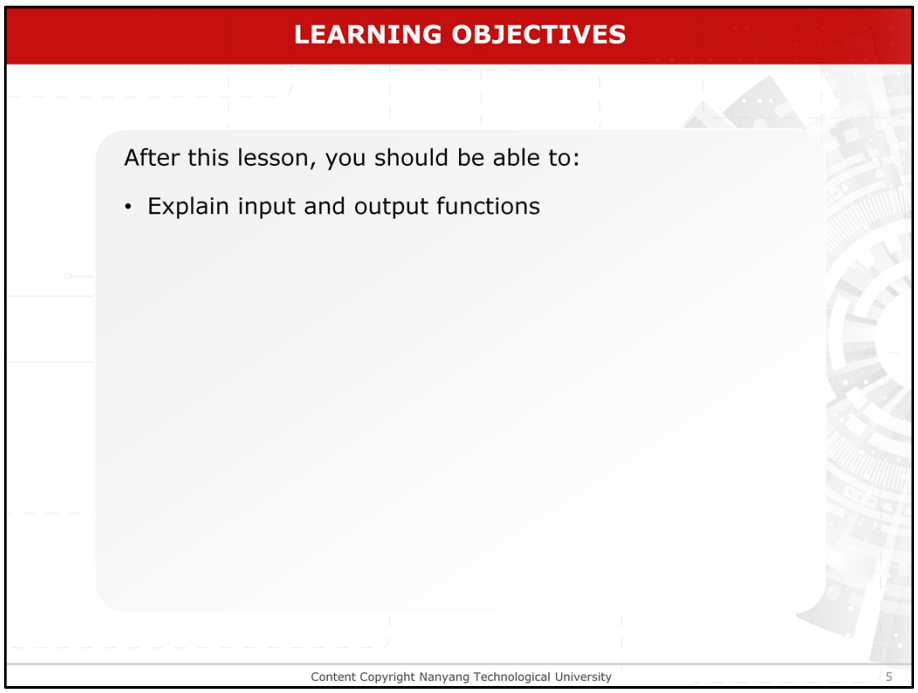
This video lesson focuses on simple input and output operations


LEARNING OBJECTIVES
Content Copyright Nanyang Technological University
3

[Pause 1 sec]

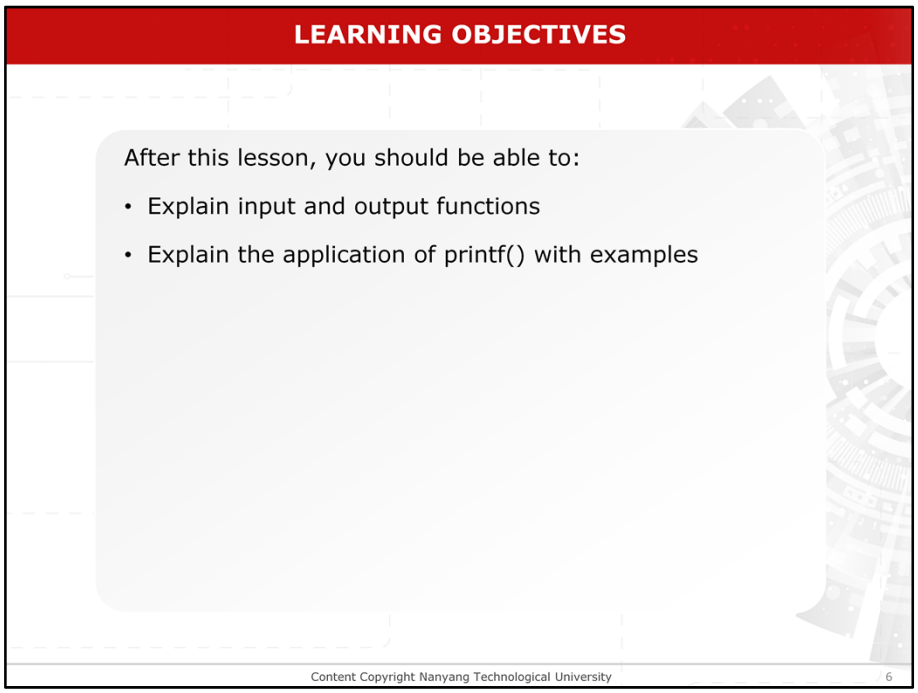


After this lesson, you should be able to:
[Pause 1 sec]



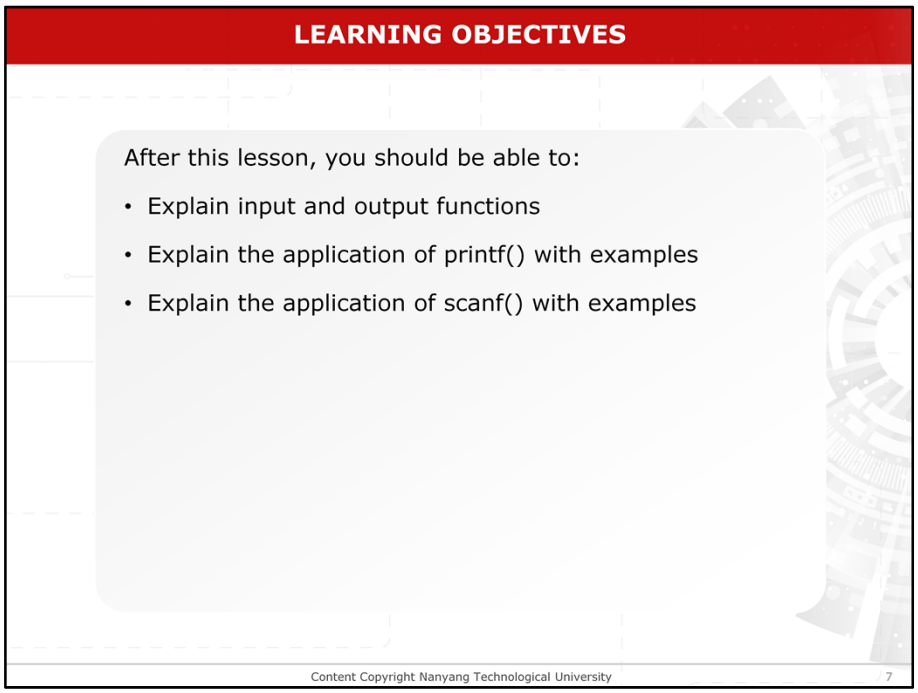
Explain input and output functions.

[Pause 1 sec]



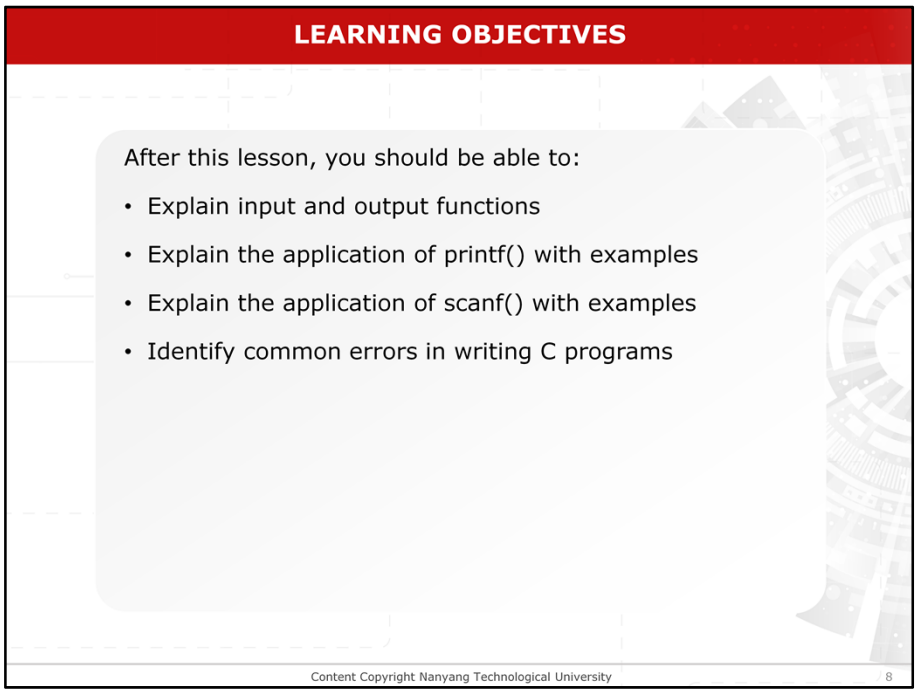
Explain the application of printf() with examples.

[Pause 1 sec]



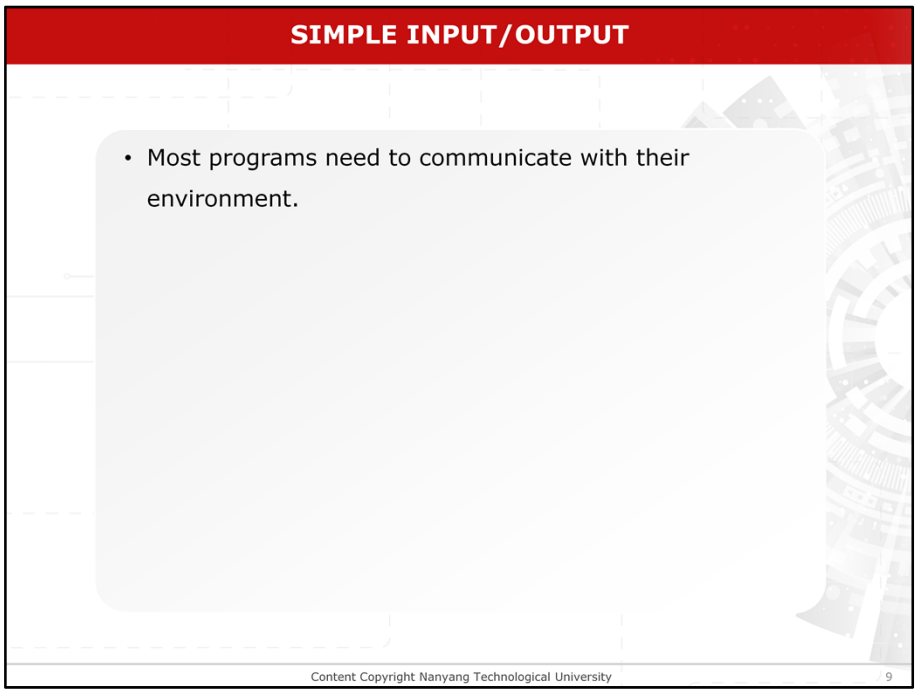
Explain the application of scanf() with examples.

[Pause 1 sec]



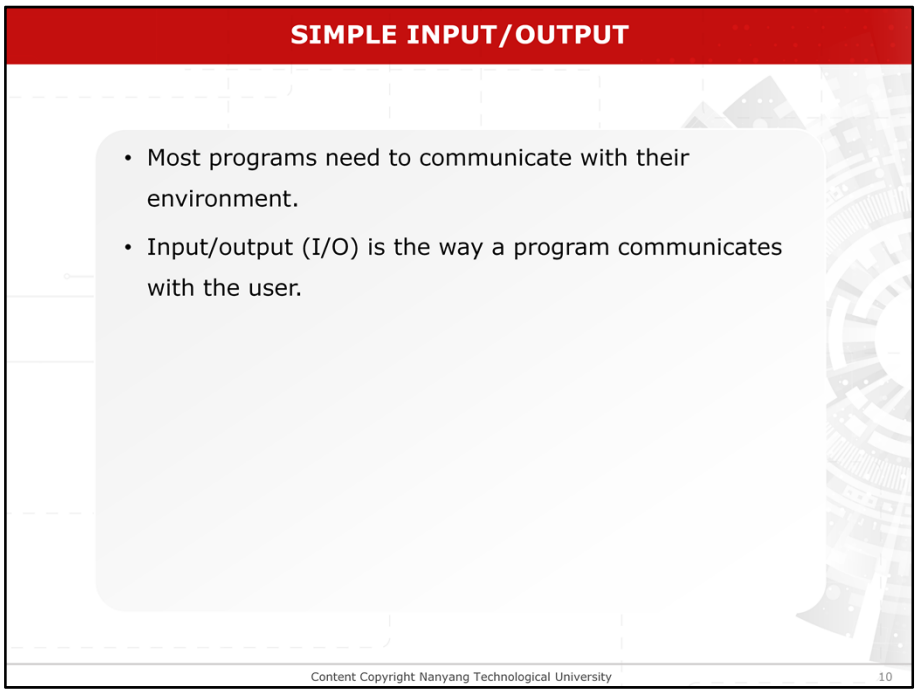
Identify common errors in writing C programs.

[Pause 1 sec]



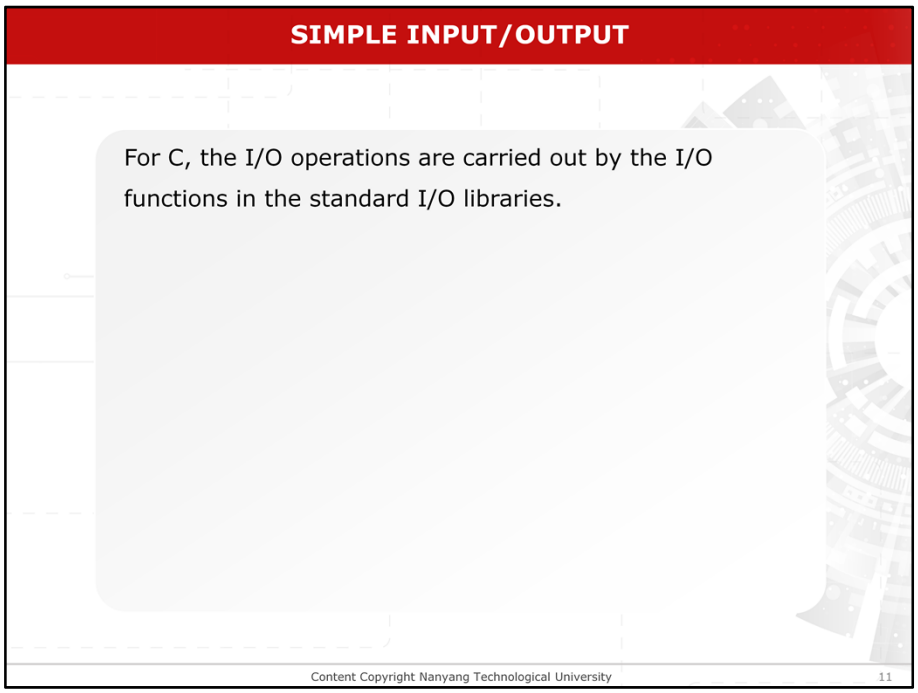
Simple Input/Output

Input/output (I/O) is the way a program communicates with the user.



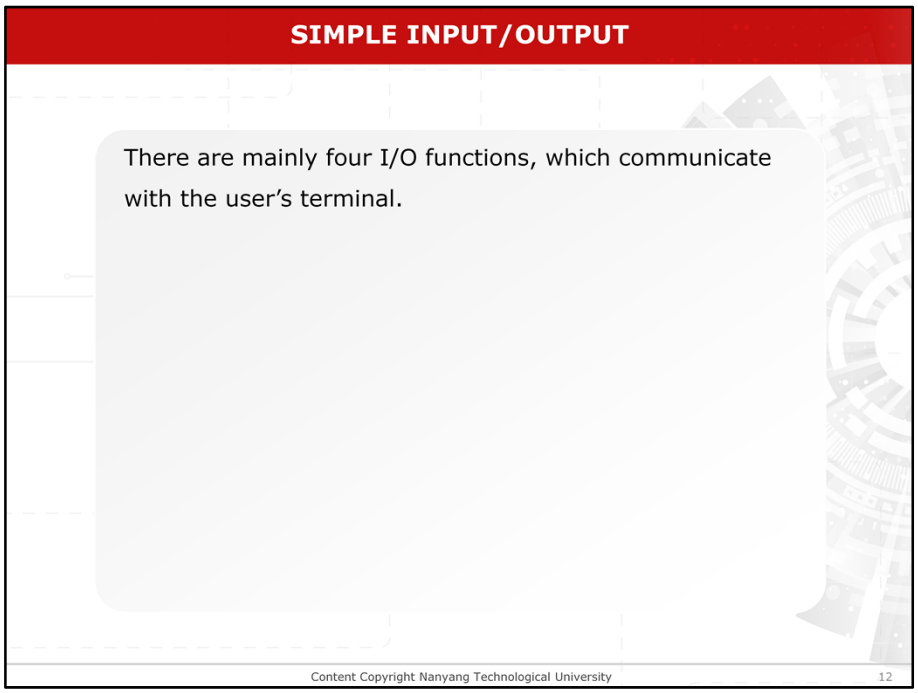
Simple Input/Output

Input/output (I/O) is the way a program communicates with the user.



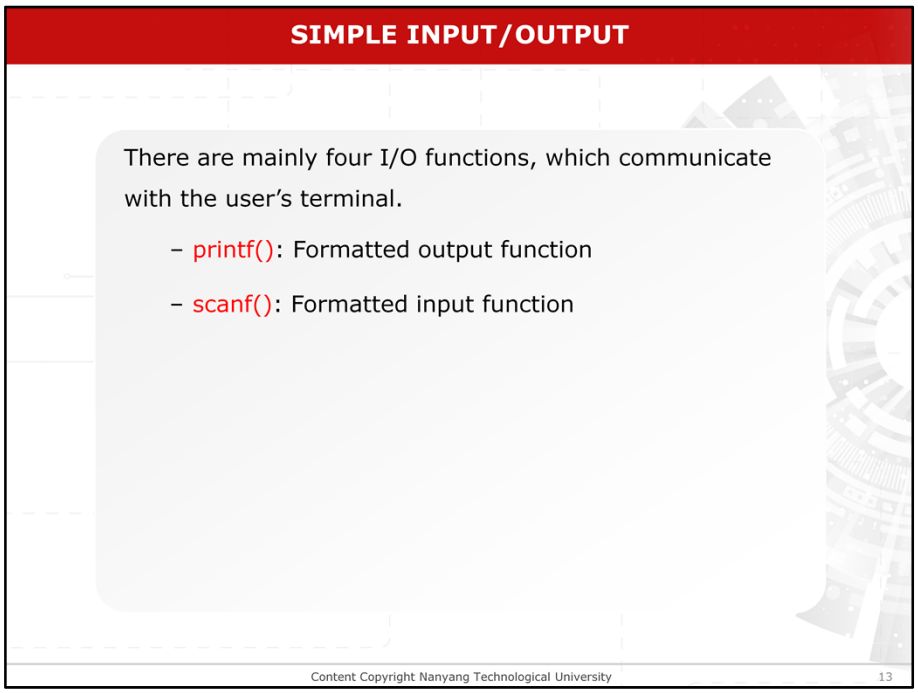
Simple Input/Output

For C, the I/O operations are carried out by the Input output functions in the standard input output libraries.

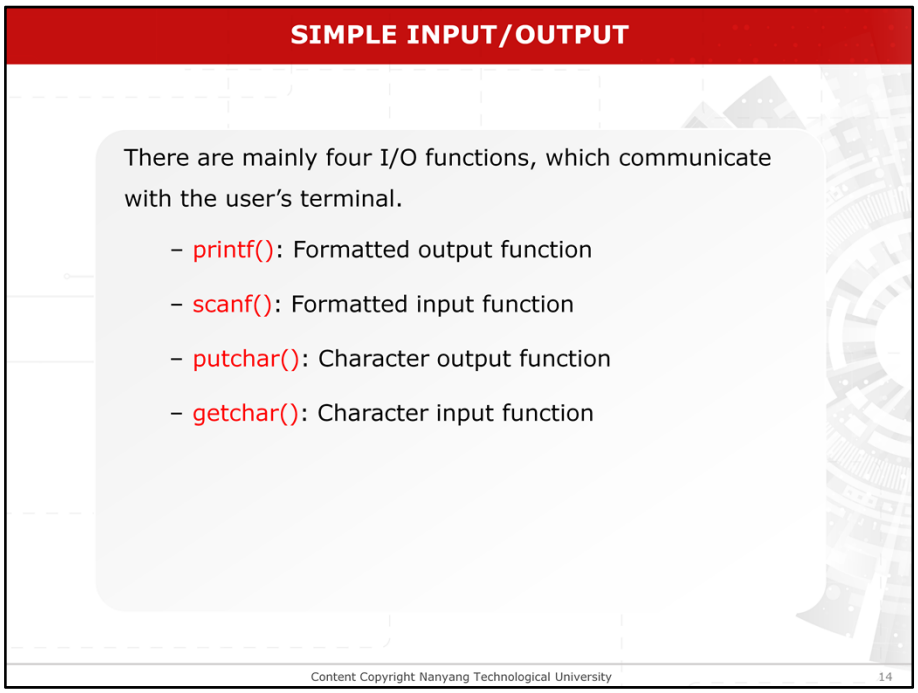


Simple Input/Output

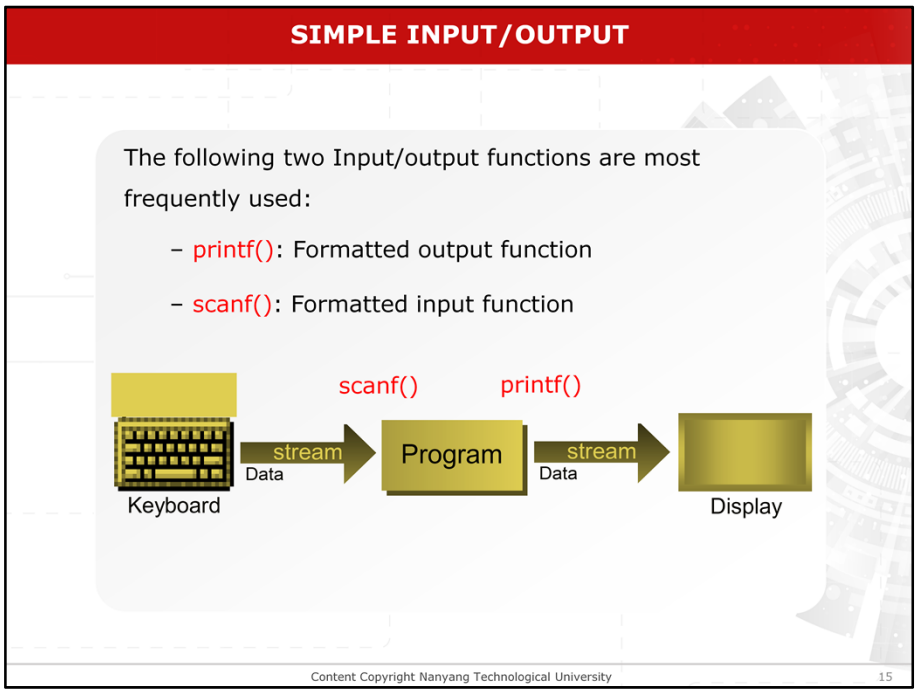
There are mainly four I/O functions, which communicate with the user's terminal.



The printf() and **scanf()** functions perform formatted Input output operations

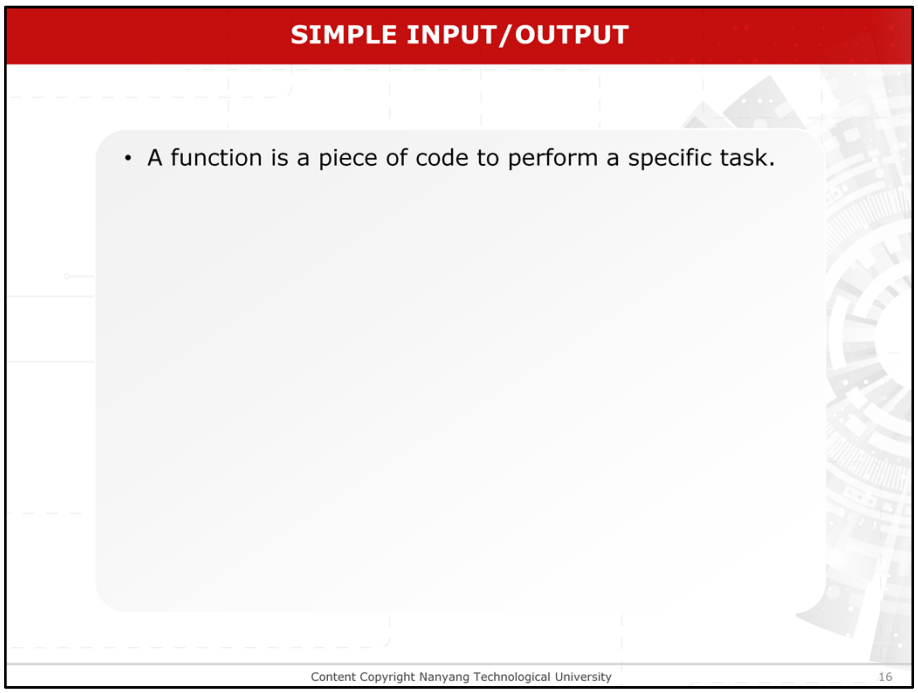


while the **putchar()** and **getchar()** functions perform character input output operations

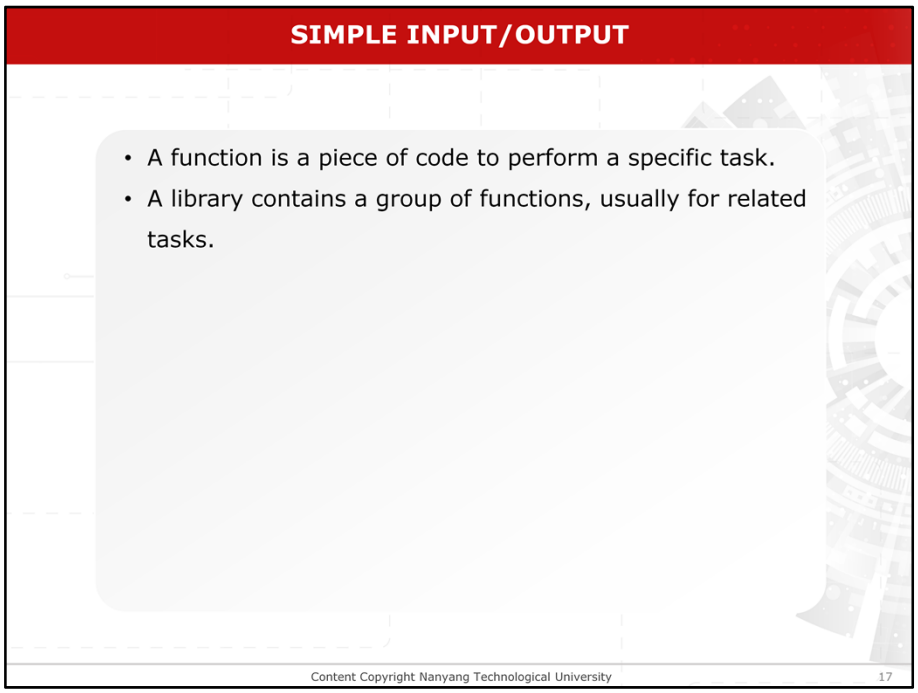


Simple Input/Output

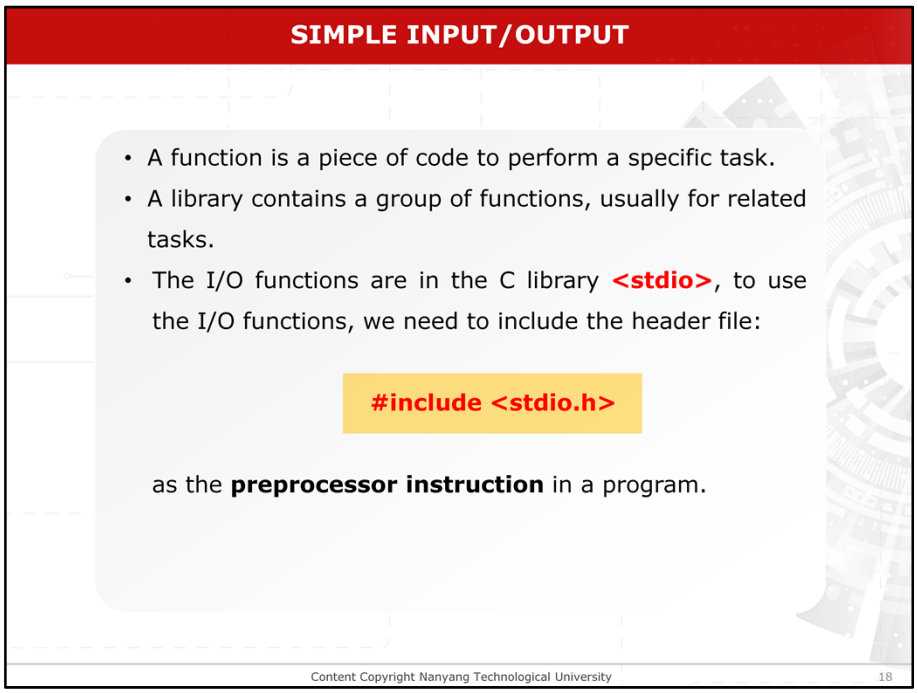
Input from the keyboard or output to the monitor screen is referred as standard input/output. There are mainly four I/O functions, which communicate with the user's terminal. The **printf()** and **scanf()** functions perform formatted I/O,



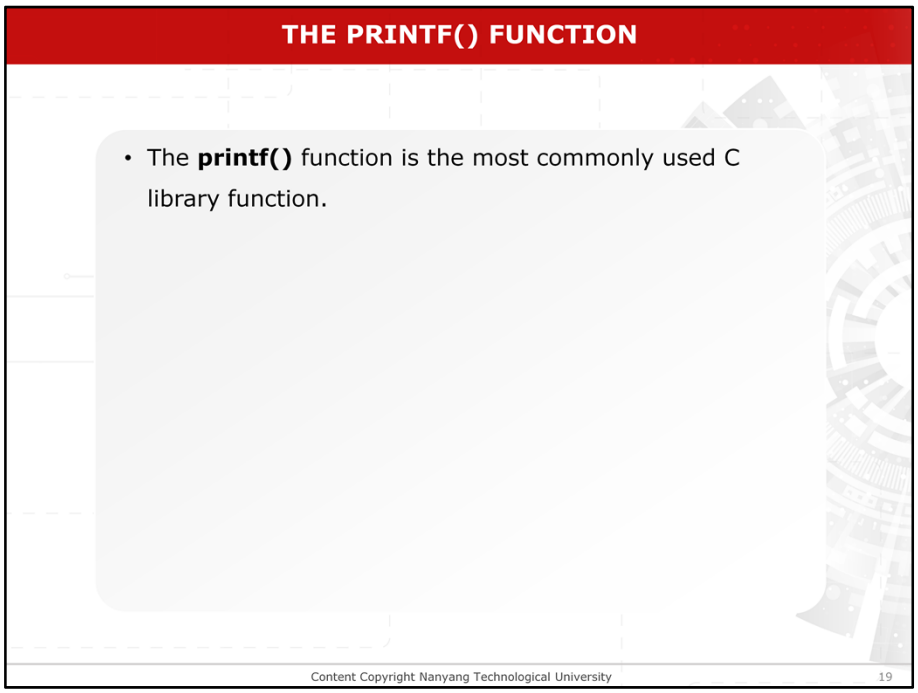
A function is a piece of code to perform a specific task



A library contains a group of functions, usually for related tasks.

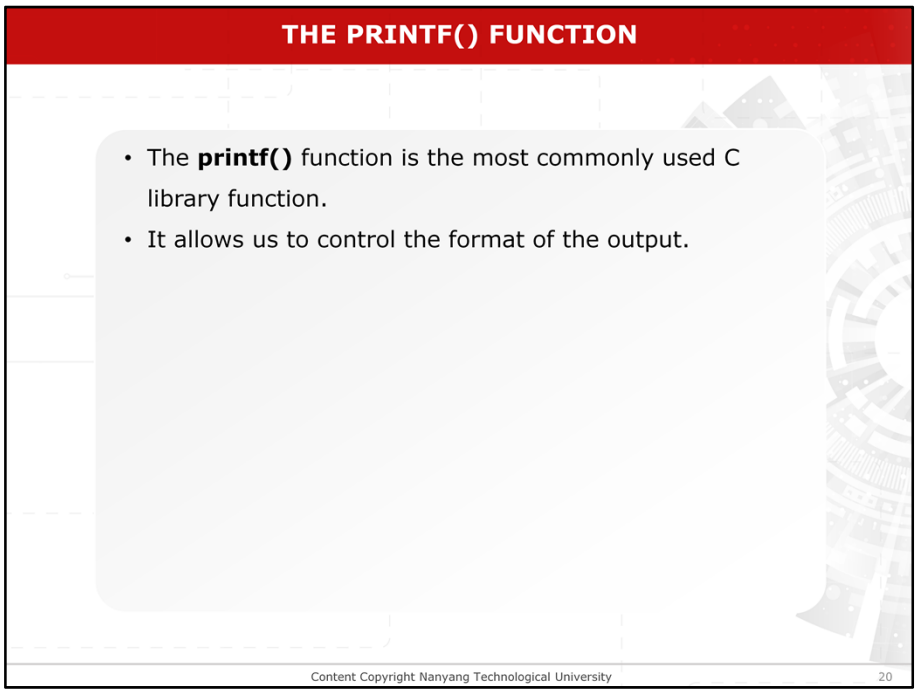


The standard I/O functions in the library *<stdio>* and the mathematical functions in the library *<math>* are two examples of library functions. To use the I/O functions in *<stdio>*, the preprocessor directive, must be included in a program.



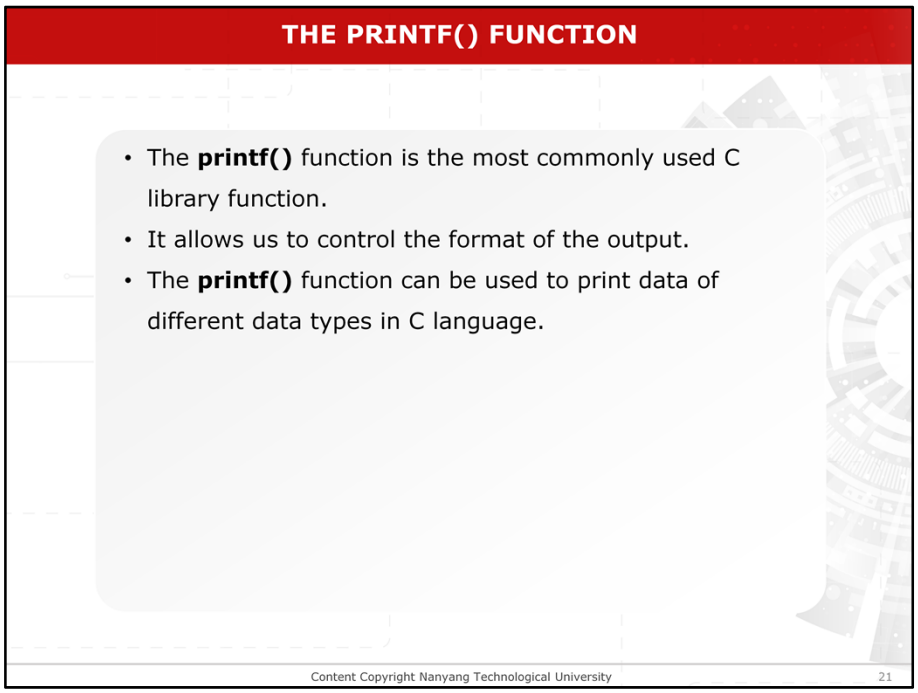
The printf() Function

The **printf()** function is the most commonly used C library function.

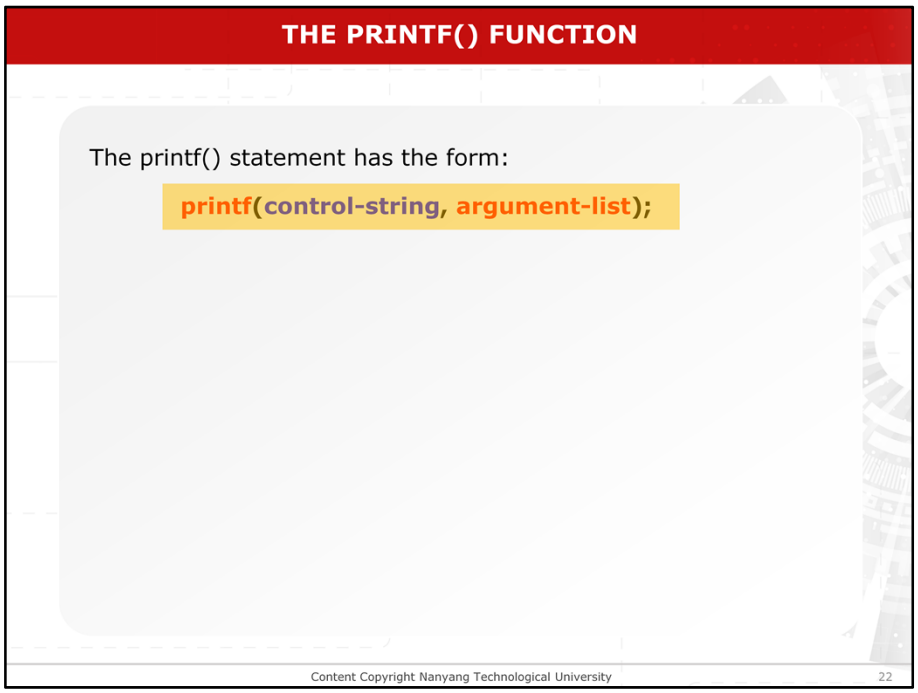


The printf() Function

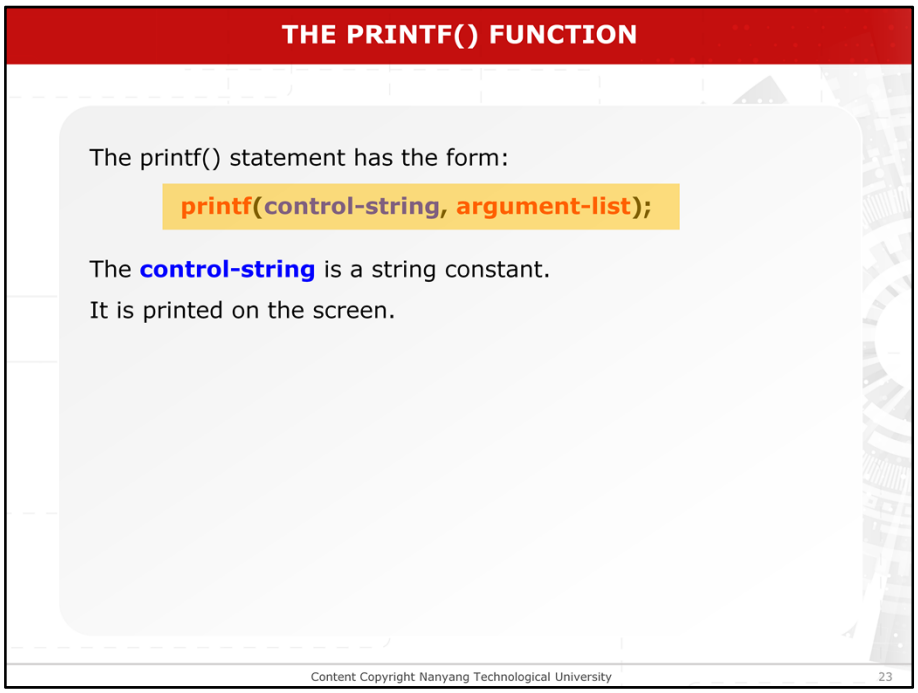
It allows us to control the format of the output.



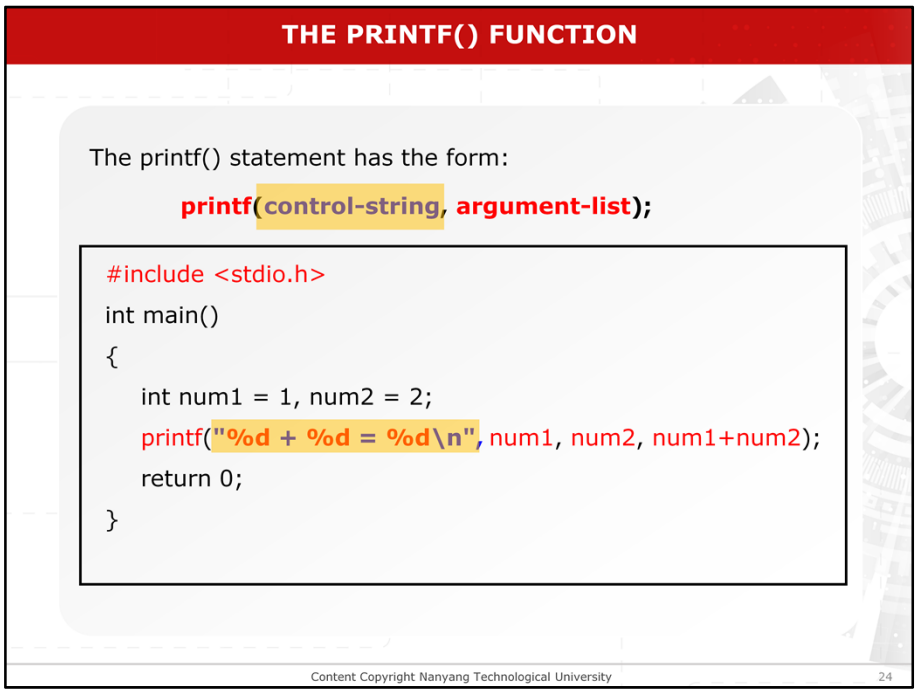
The **printf()** function can be used to print data of different data types in C language.



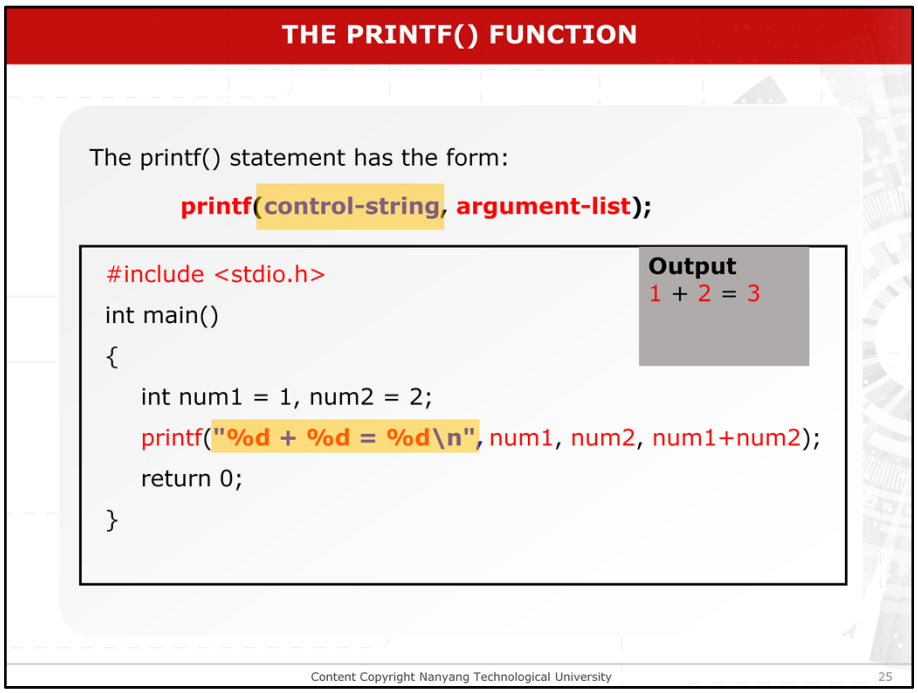
A **printf()** statement has the format as shown.



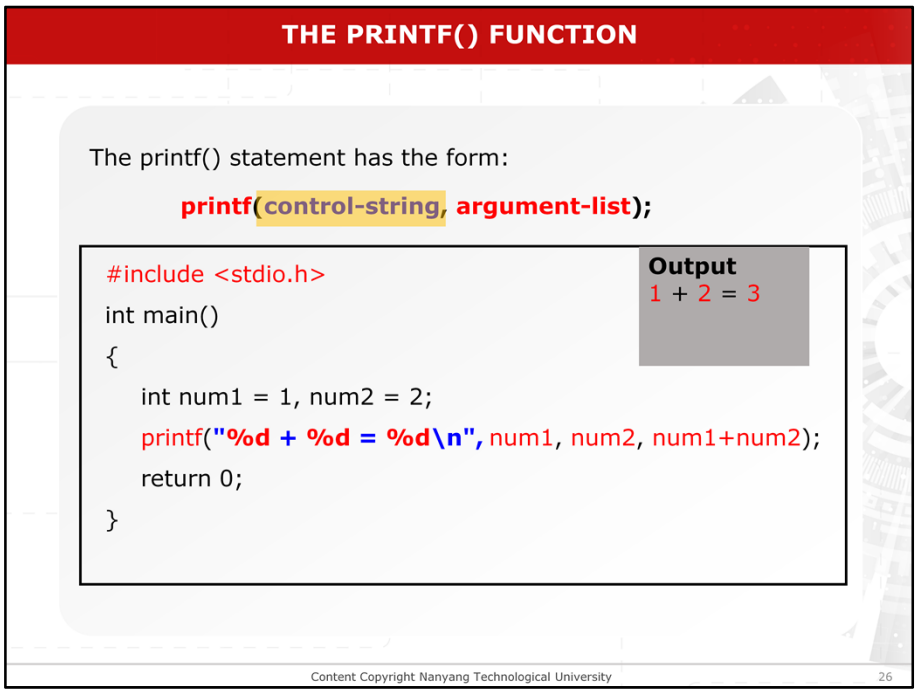
The **control-string** is a string constant. It is printed on the screen.



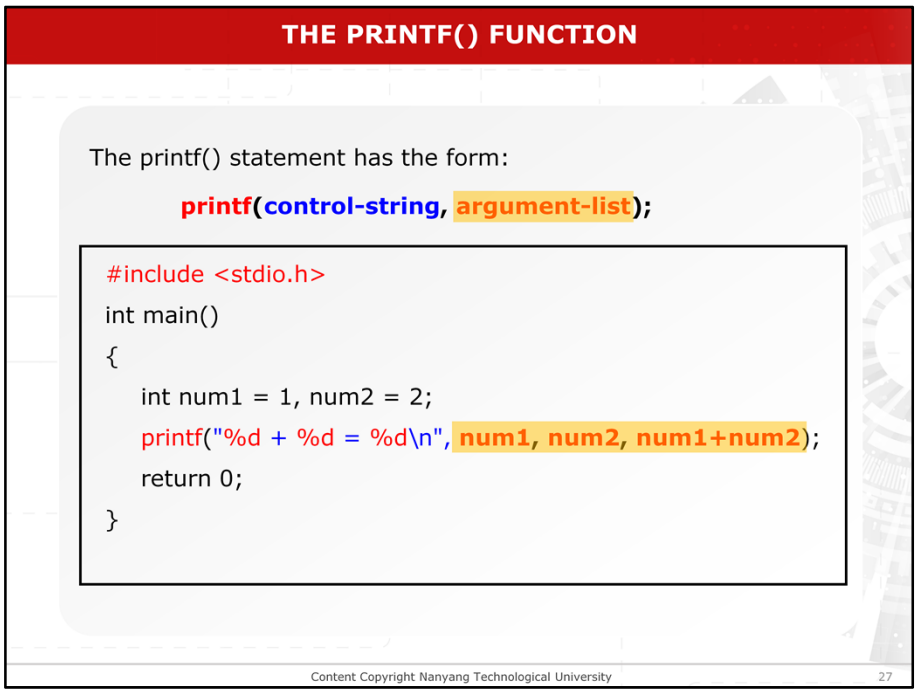
The **control-string** which is enclosed in double quotation marks is a string constant.



Two types of information are specified in the **control-string**. It comprises the characters that are to be printed and the **conversion specifications** (or **format specifiers**). In the program, **%d** is the conversion specification. It defines the ways the items are displayed on the screen. Conversion specifications can be placed anywhere within the **control-string**. Three conversion specifications with **%d** are specified in the program.



The printf statement has the form as shown here.



Argument list:
The **argument-list** contains a list of items such as item1, item2, ..., etc. to be printed. They contain values to be substituted into places held by the conversion specifications in the **control-string**.
An **item** can be a constant, a variable or an expression such as **num1+num2**.

PRINTF(): ARGUMENT LIST

The printf() statement has the form:

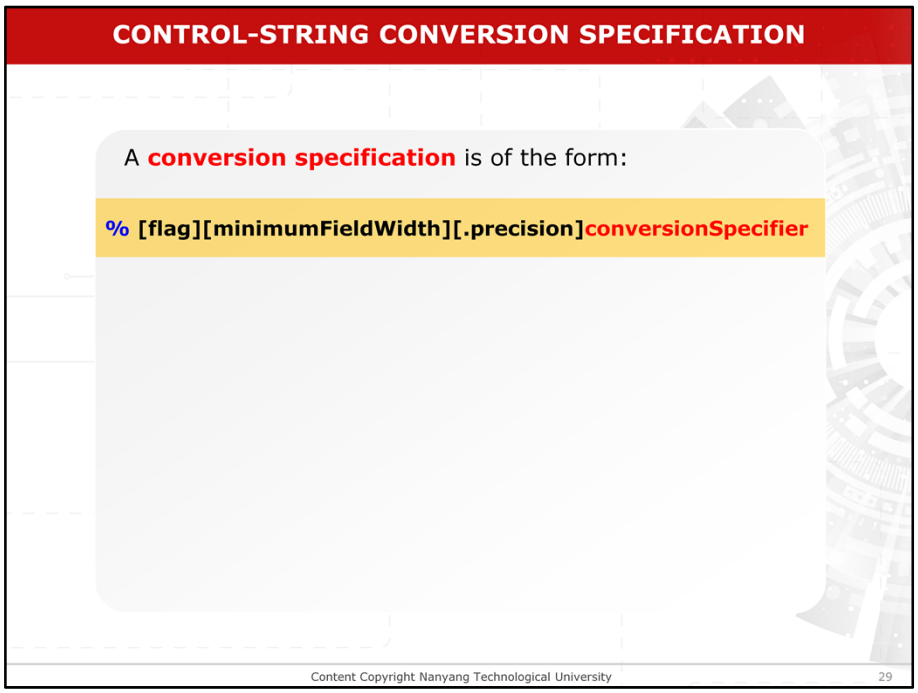
printf(control-string, argument-list);

```
#include <stdio.h>
int main()
{
    int num1 = 1, num2 = 2;
    printf("%d + %d = %d\n", num1, num2, num1+num2);
    return 0;
}
```

Output

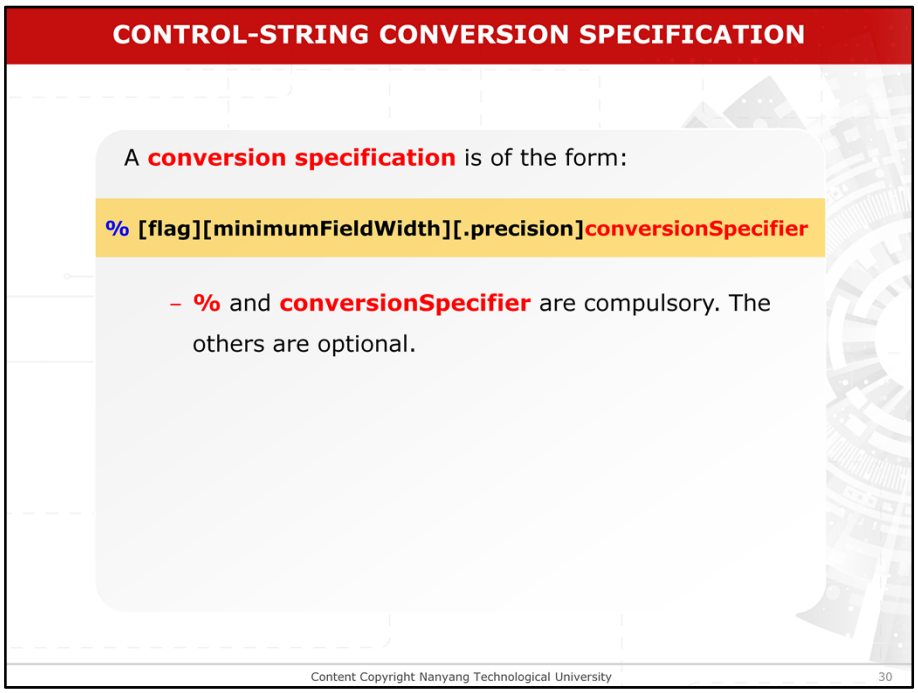
```
1 + 2 = 3
```


Note that the number of items must be the same as the number of conversion specifications, and the type of the item must match with the **conversion specifier**.



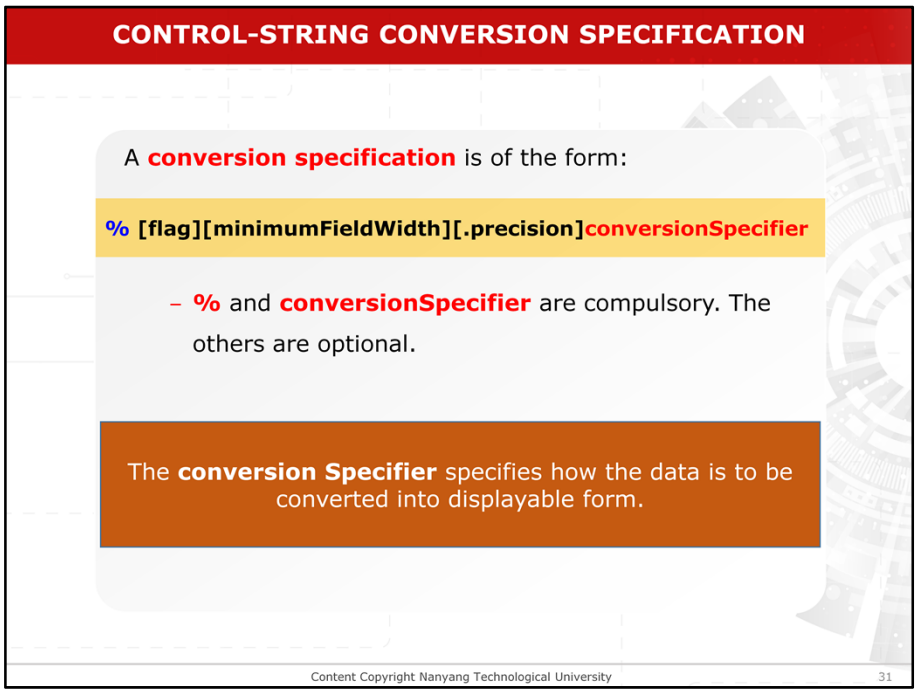
Conversion Specification

The format of a conversion specification is shown here.



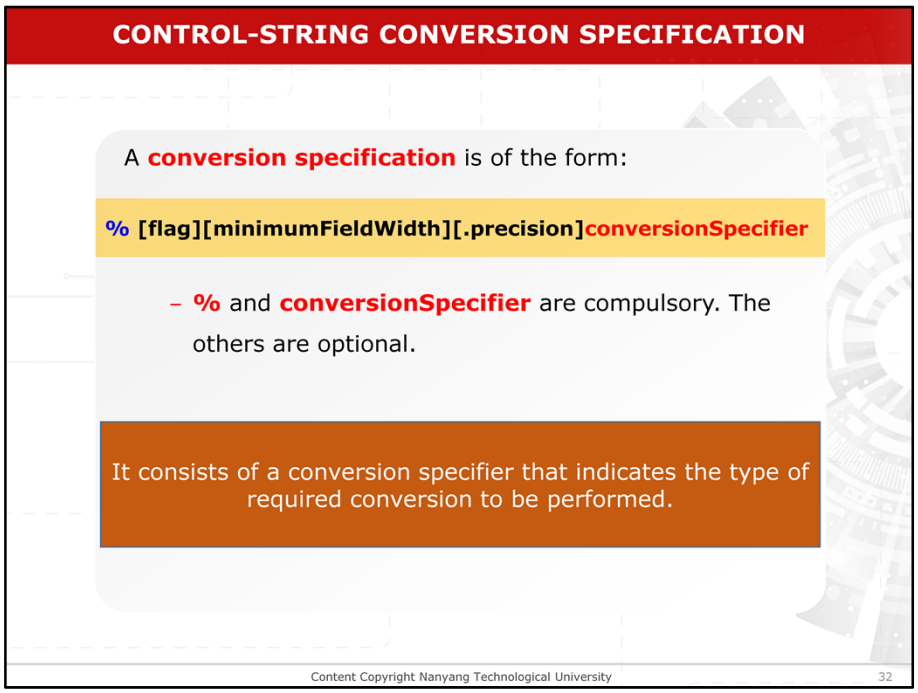
Conversion Specification

The format of a conversion specification is shown here.



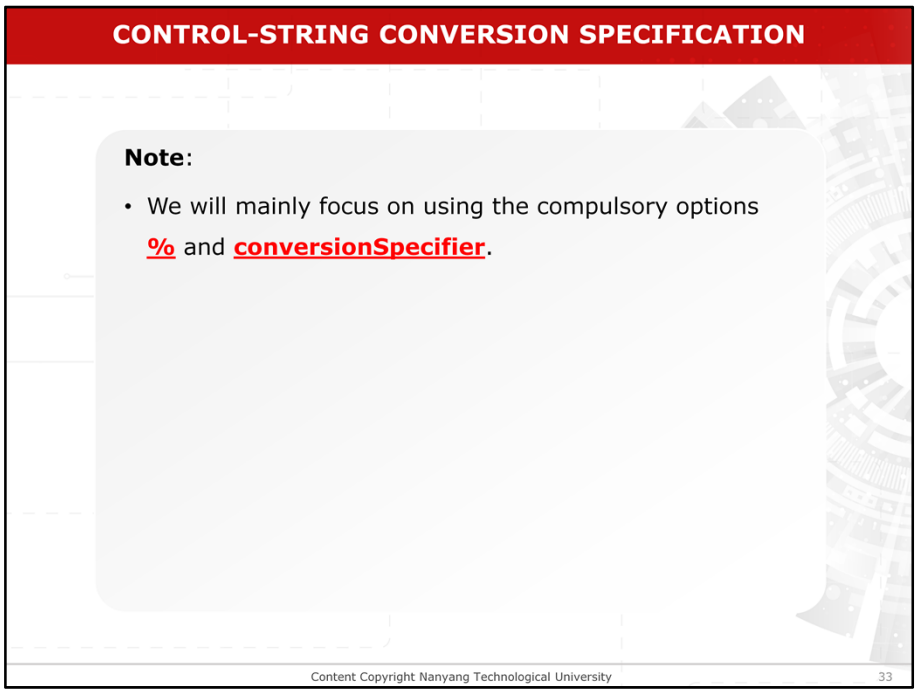
Conversion Specification

The format of a conversion specification is shown here.



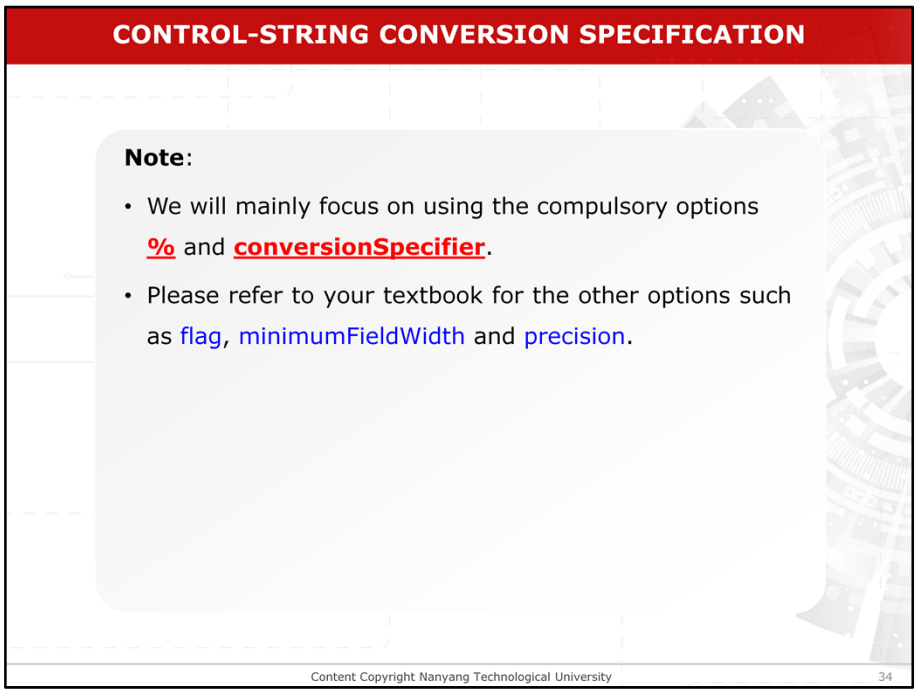
Conversion Specification

The format of a conversion specification is shown here.



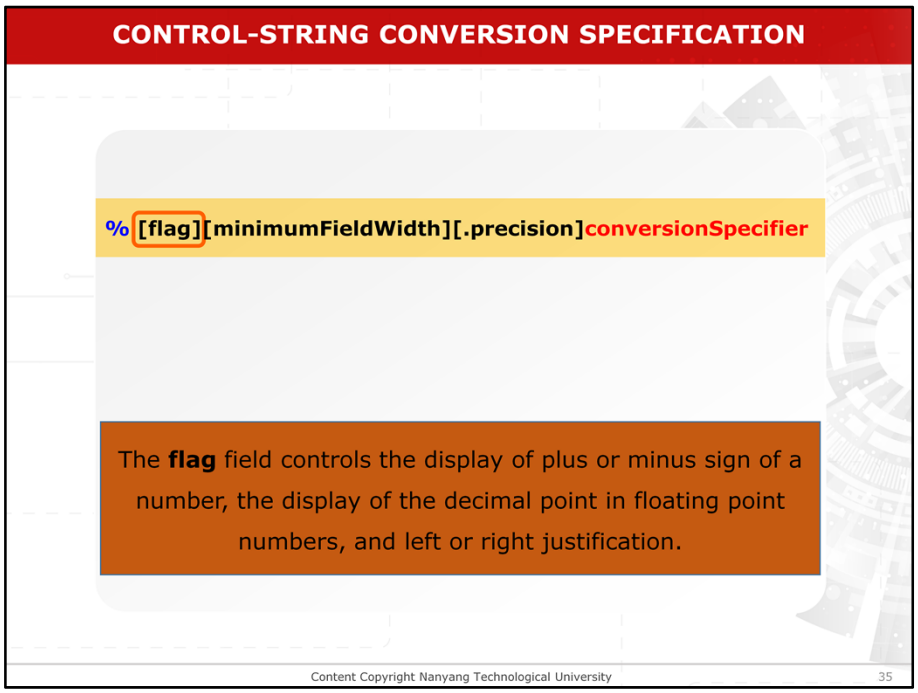
Conversion Specification

Note that in this course, we focus on the compulsory options % and conversionSpecifier.



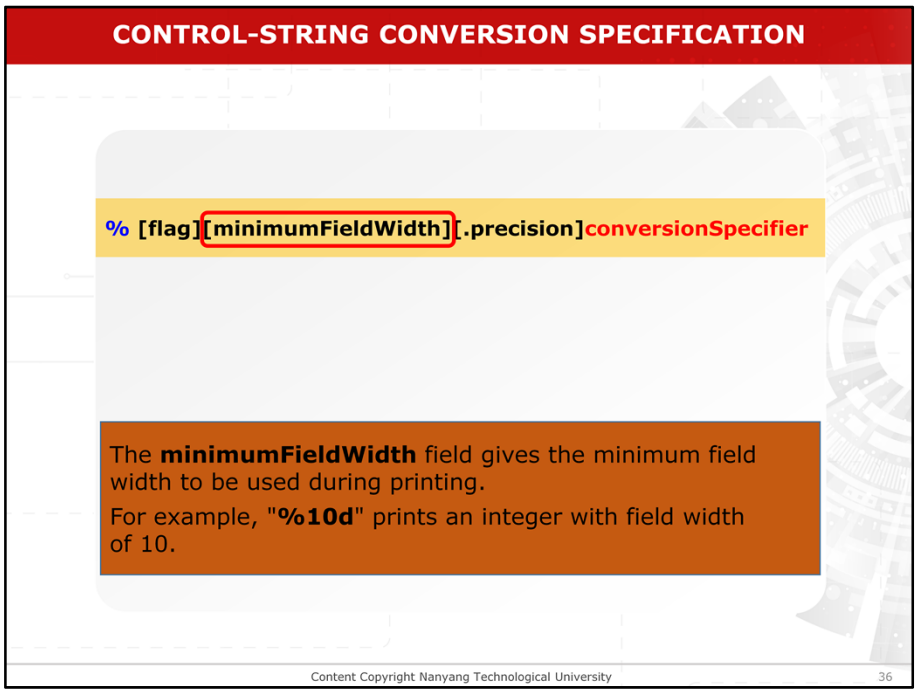
Conversion Specification

Note that in this course, we mainly focus on the compulsory options % and conversionSpecifier.

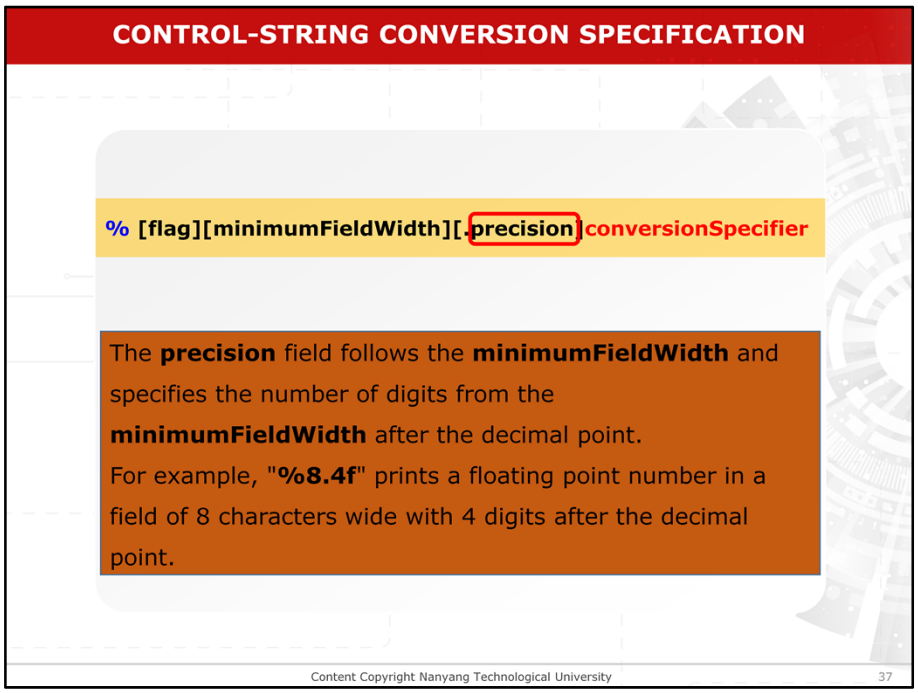


The options provide the additional control on how to print a value as follows:

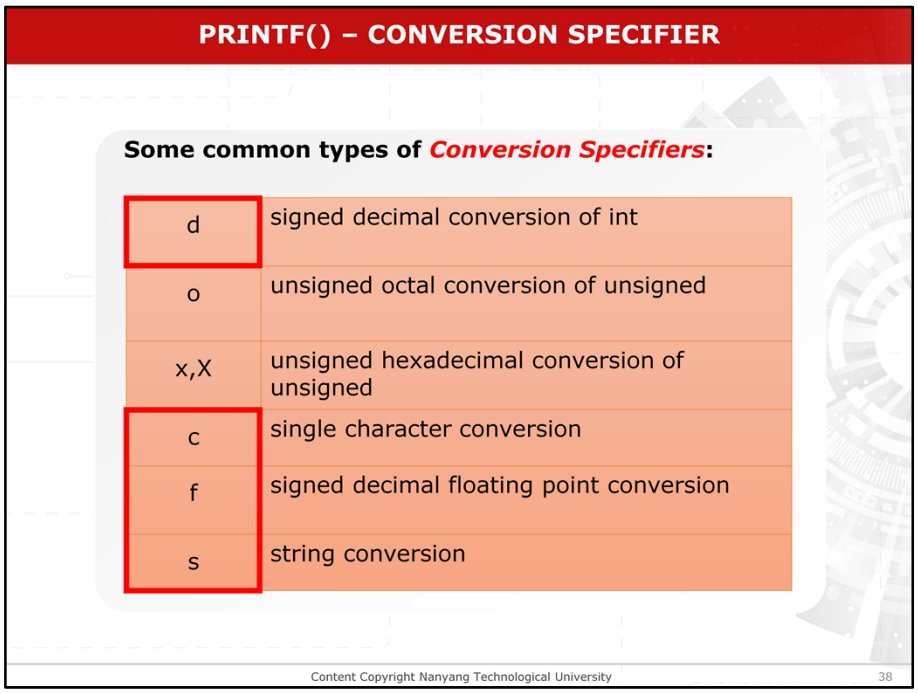
controls the display of plus or minus sign of a number, the display of the decimal point in floating point numbers, and left or right justification



The **minimumFieldWidth** field gives the minimum field width to be used during printing.

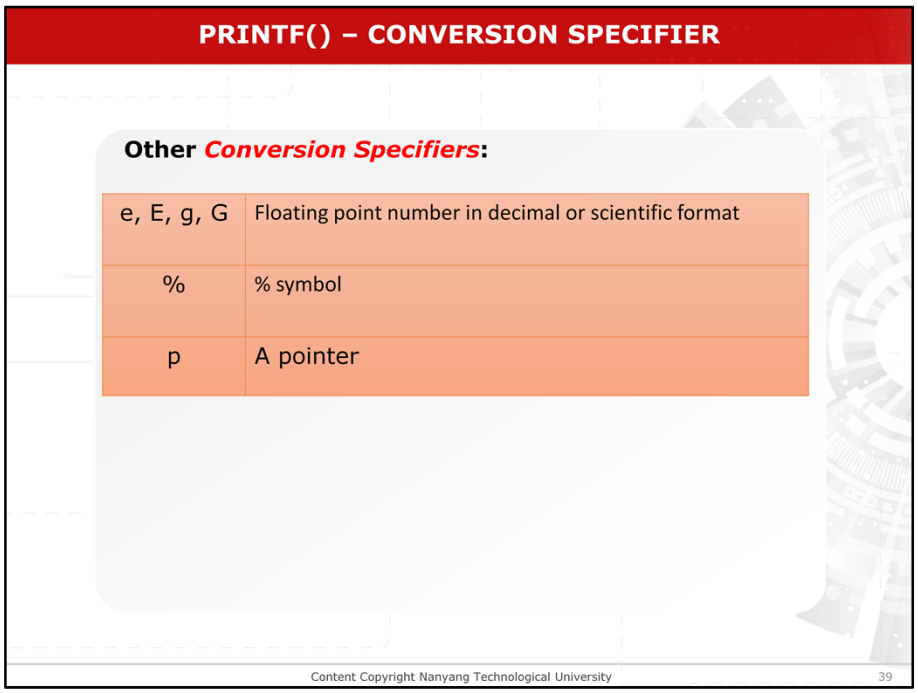


The **precision** field follows the **minimumFieldWidth** and specifies the number of digits from the **minimumFieldWidth** after the decimal point.



Conversion Specifier

The most common conversion specifiers are **d** for decimal integers, **c** for characters, **f** for floating point numbers, and **s** for strings.



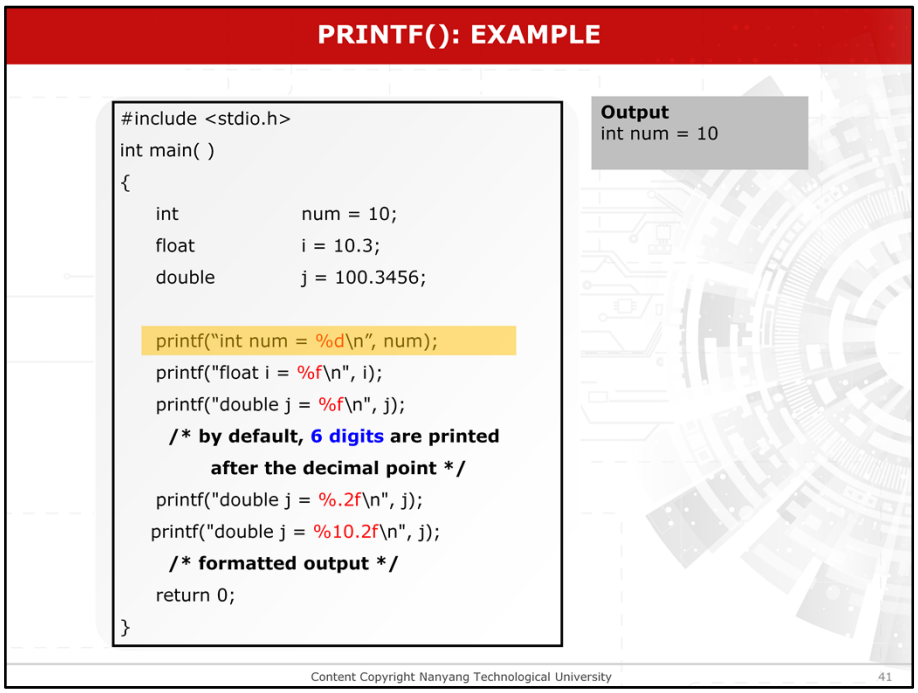
Other conversion specifiers include **e**, **E**, **p**, **g**, **G** and **%**:

- **e, E, g, G** - floating point number in decimal or scientific format;
- **p** - a pointer;
- **% -** the % symbol.

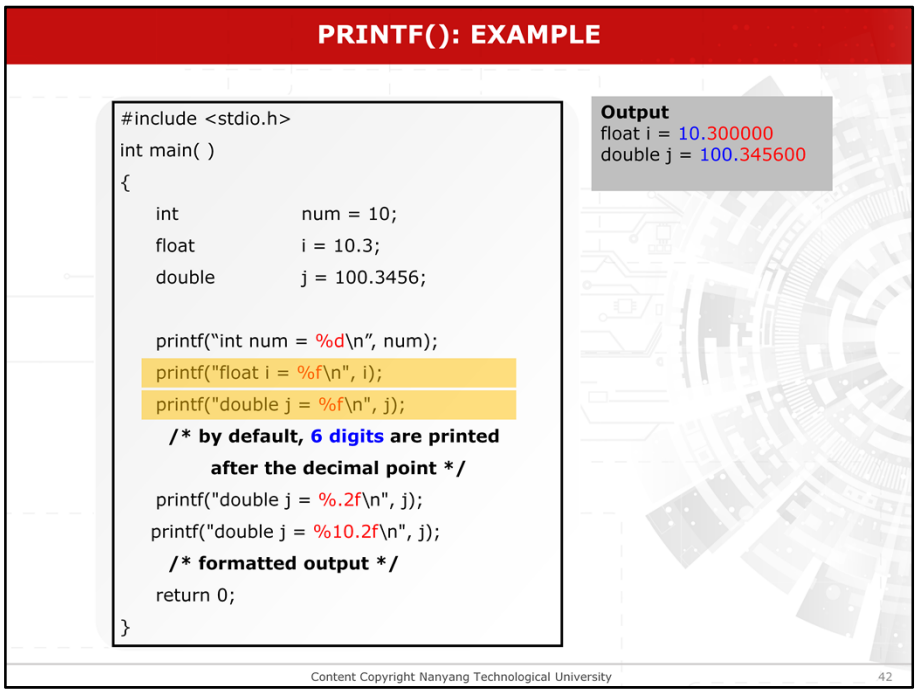
PRINTF(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    int             num = 10;
    float           i = 10.3;
    double          j = 100.3456;

    printf("int num = %d\n", num);
    printf("float i = %f\n", i);
    printf("double j = %f\n", j);
      /* by default, 6 digits are printed
           after the decimal point */
    printf("double j = %.2f\n", j);
    printf("double j = %10.2f\n", j);
      /* formatted output */
    return 0;
}
```


In the program, it prints an integer using conversion specification **%d** and two floating point numbers using the conversion specification **%f** with different options



The first printf() statement prints an integer.



The second and third **printf()** statements print the numbers using **%f**. The default precision of 6 is used, i.e. six digits are printed after the decimal point.

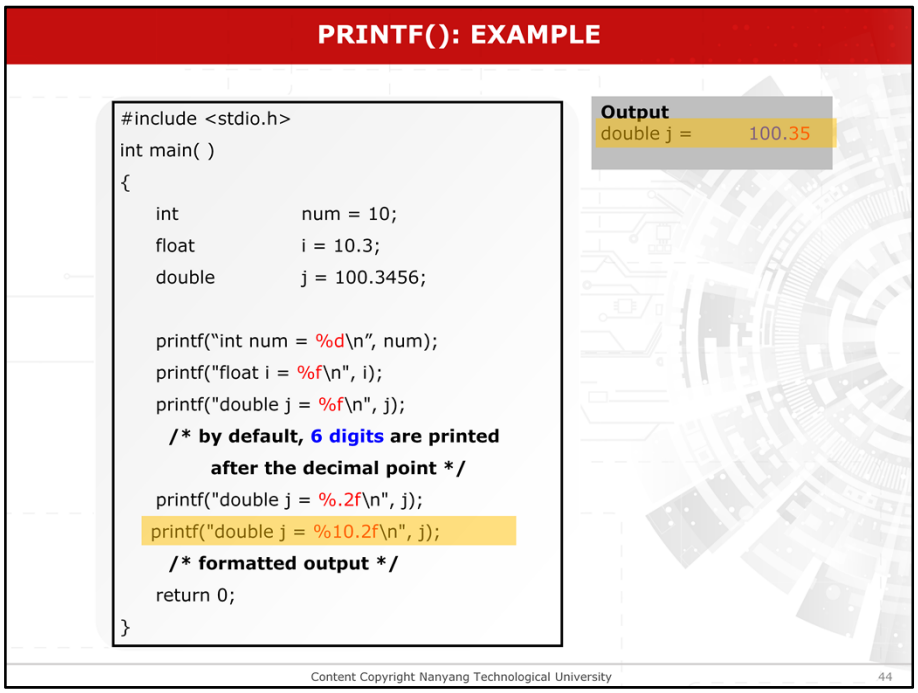
PRINTF(): EXAMPLE

```
#include <stdio.h>
int main( )
{
    int             num = 10;
    float           i = 10.3;
    double          j = 100.3456;

    printf("int num = %d\n", num);
    printf("float i = %f\n", i);
    printf("double j = %f\n", j);
     /* by default, 6 digits are printed
          after the decimal point */
    printf("double j = %.2f\n", j);
    printf("double j = %10.2f\n", j);
     /* formatted output */
    return 0;
}
```

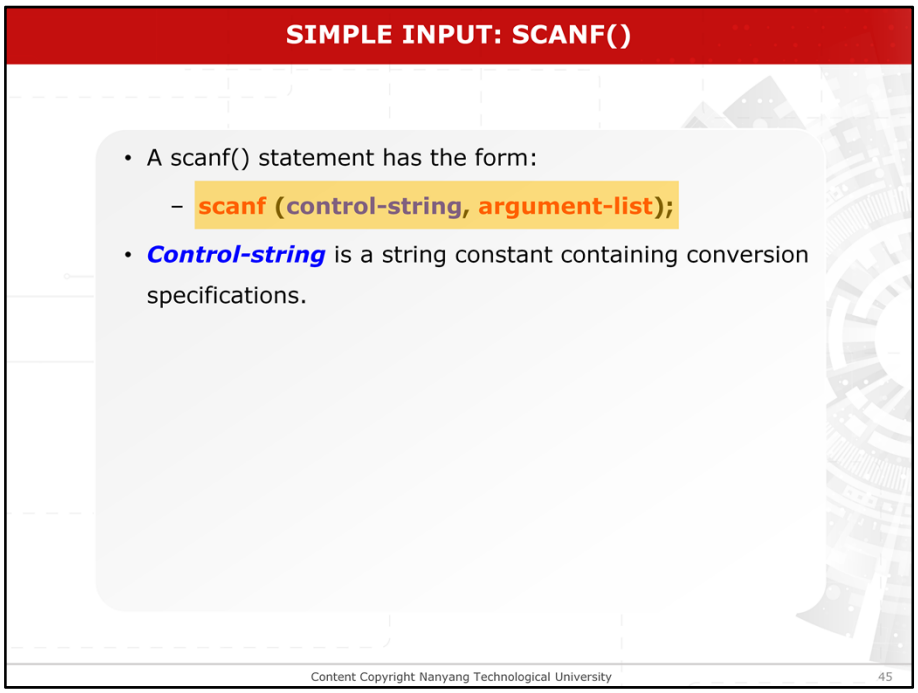
Output
double j = 100.35

 43

The fourth **printf()** statement print the numbers with precision 2, i.e. only two digits are printed after the decimal point.



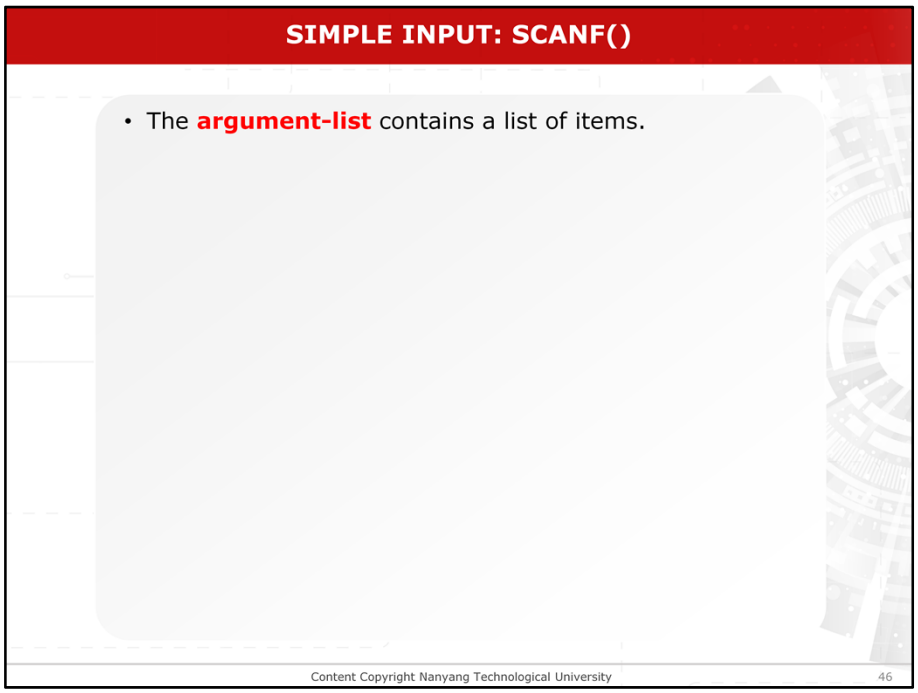
Similarly in the fifth **printf()** statement, the field width is limited to 10 with precision 2. The floating point numbers to be printed are right-justified in the field.



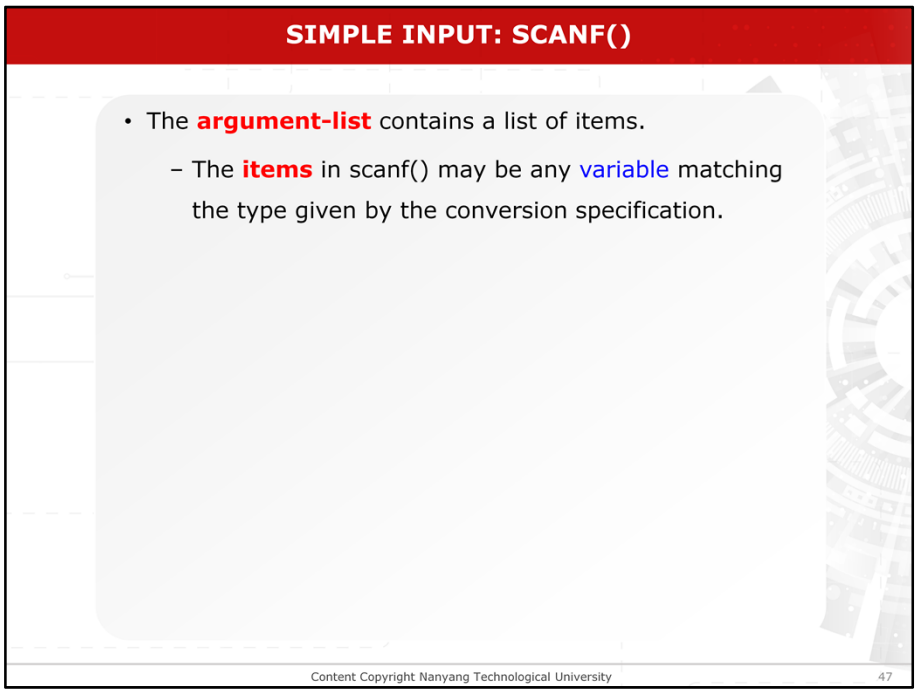
The scanf() Function

A **scanf()** statement has the following format:

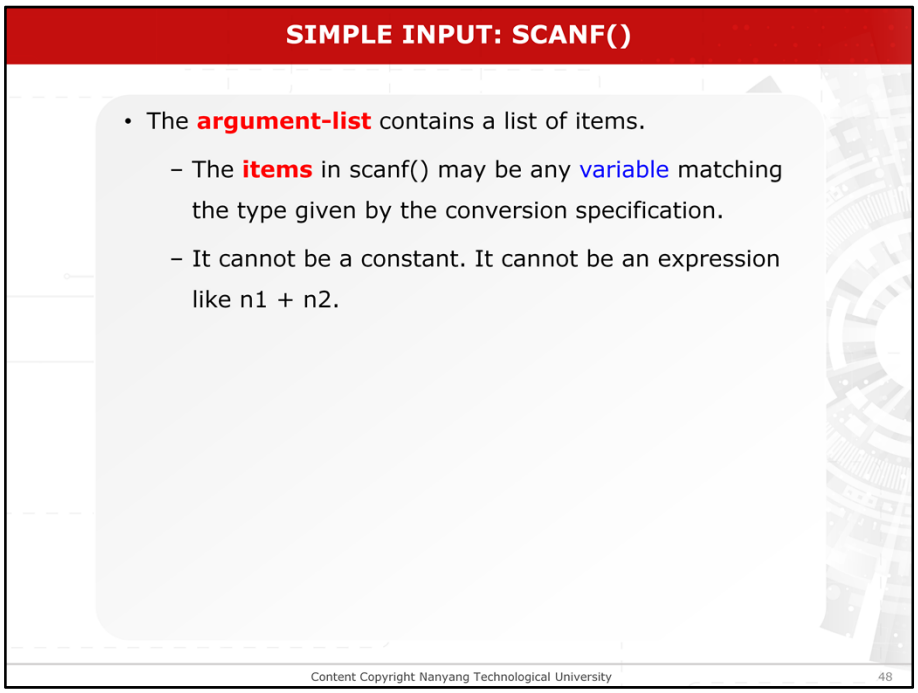
where control-string is a string constant containing conversion specifications.



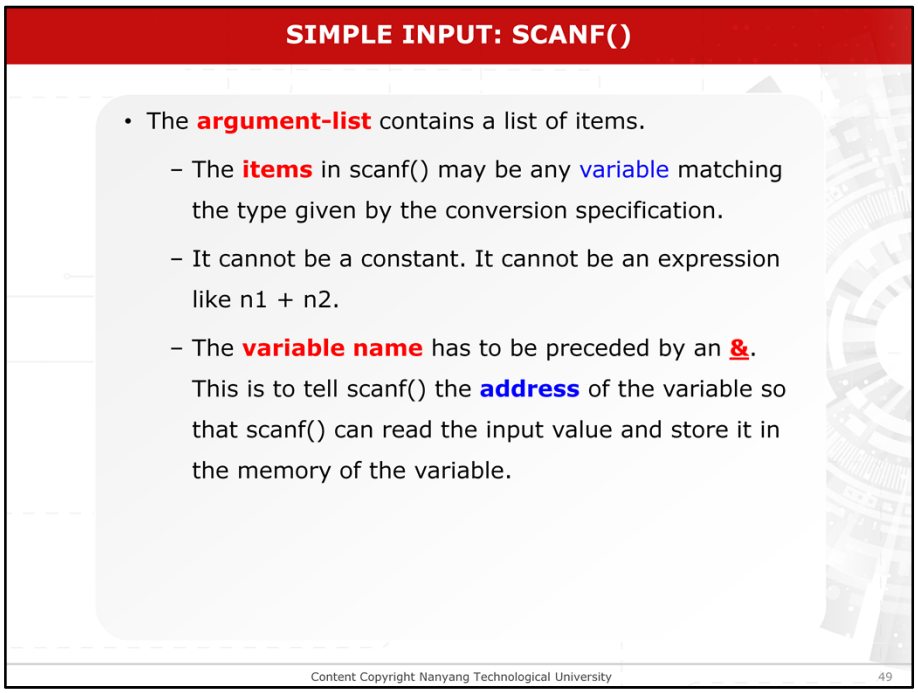
The **argument-list** contains the addresses of a list of input items



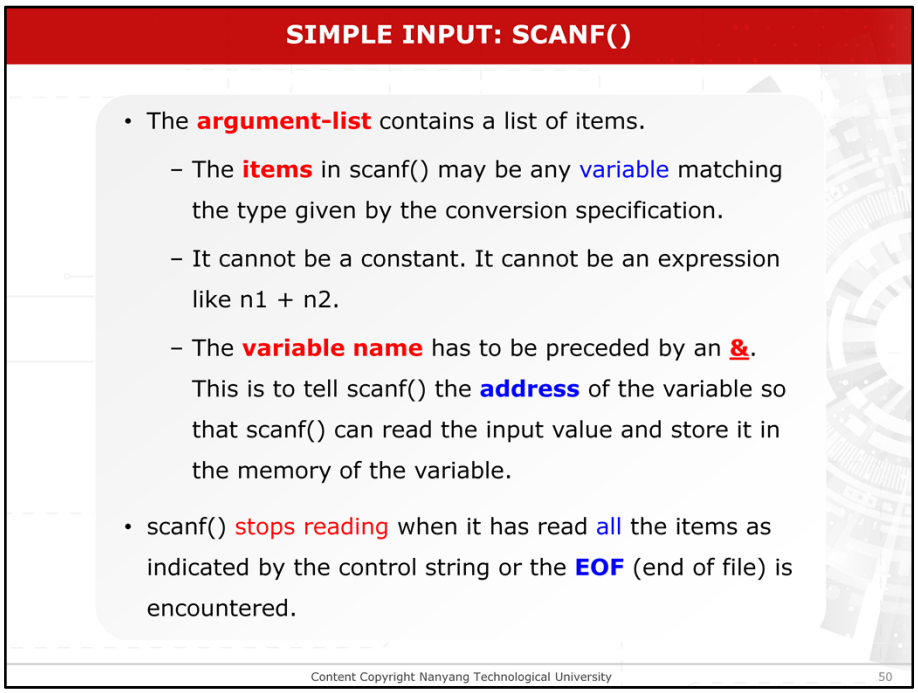
that may be any variables matching the type given by the conversion specification.



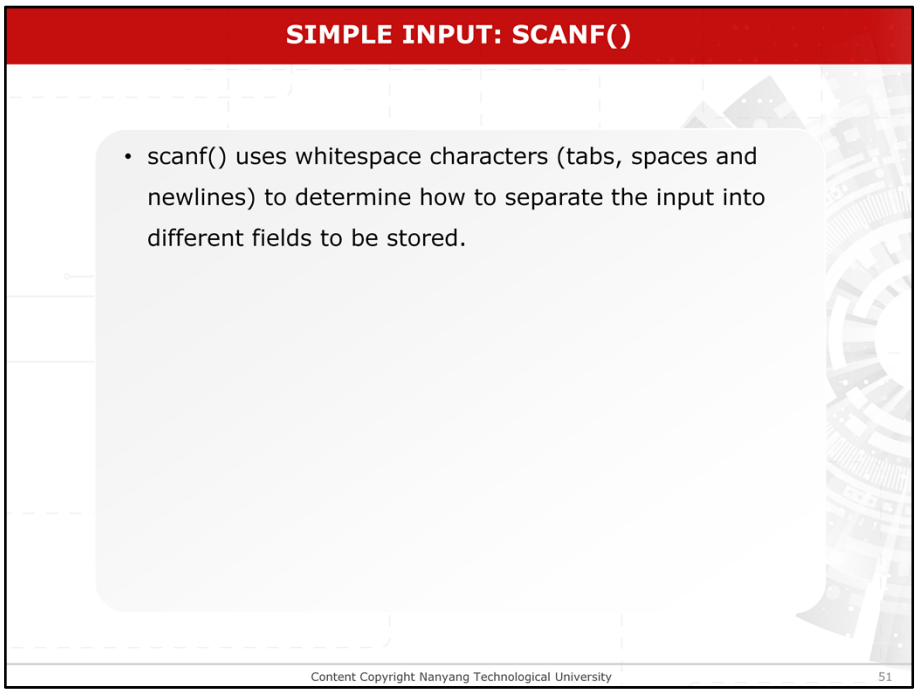
The input items cannot be a constant, or an expression such as **num1+num2**.



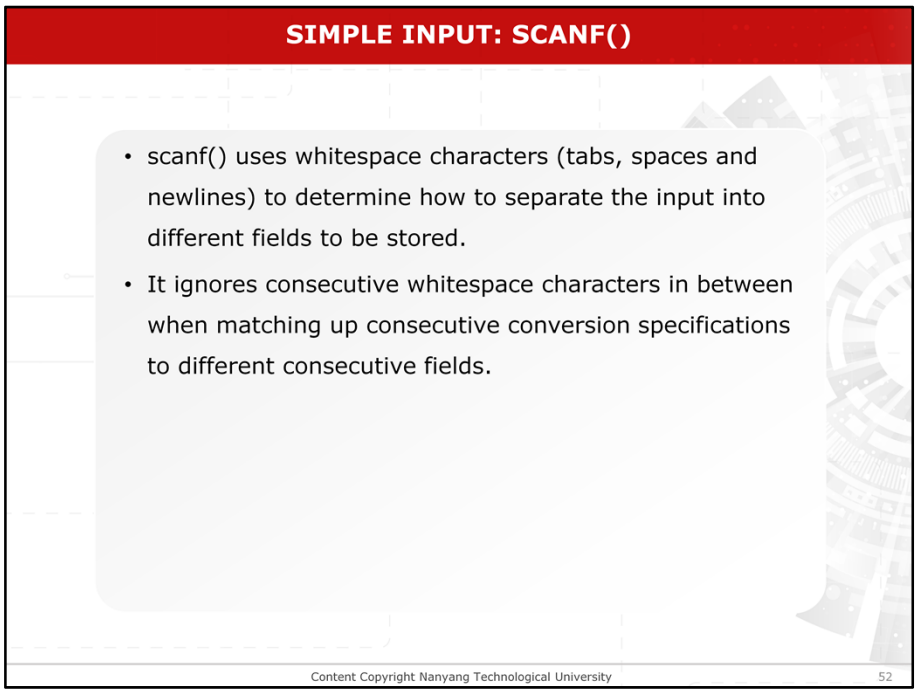
The variable name has to be preceded by an address operator **&**. This is to tell **scanf()** the address of the variable so that **scanf()** can read the input value and store it in the memory of the variable.



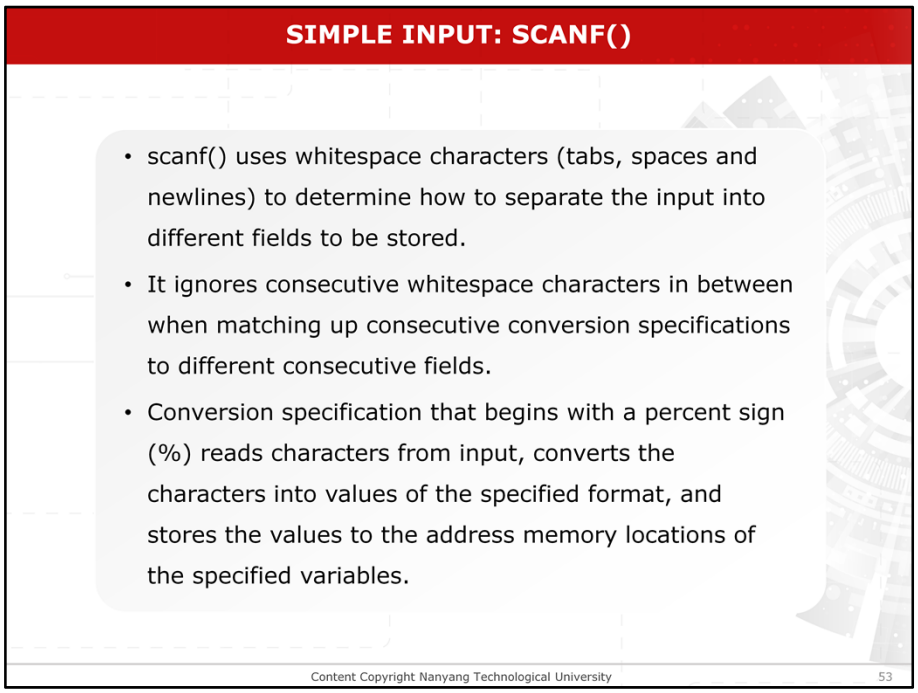
The **scanf()** statement reads data from the keyboard. It stops reading when it has read all the items as indicated by the **control-string** or the **EOF** (end-of-file) is encountered.



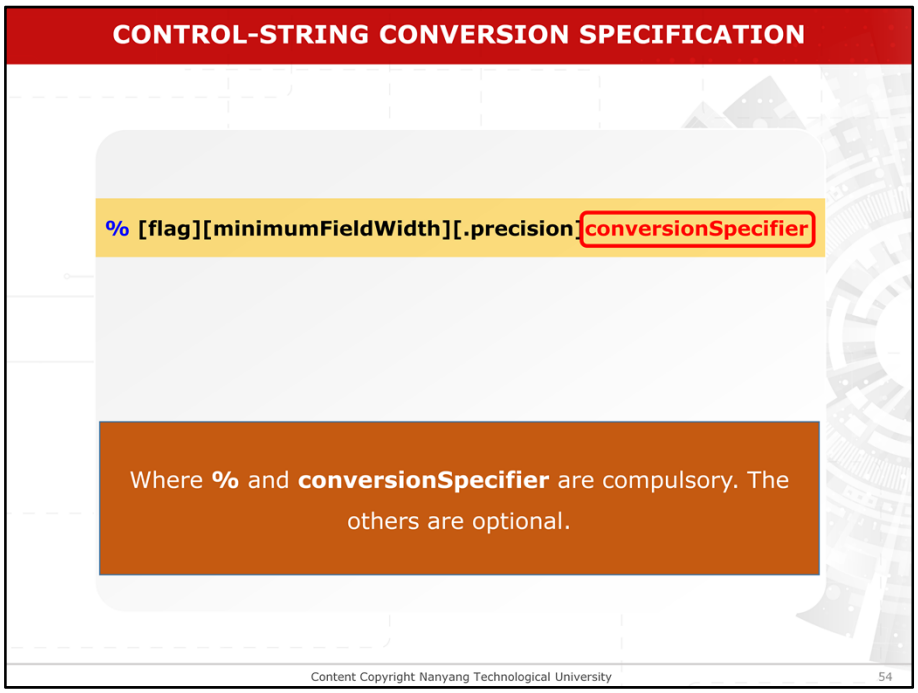
scanf() uses whitespace characters (tabs, spaces and newlines) to determine how to separate the input into different fields to be stored.



scanf() uses whitespace characters (tabs, spaces and newlines) to determine how to separate the input into different fields to be stored.

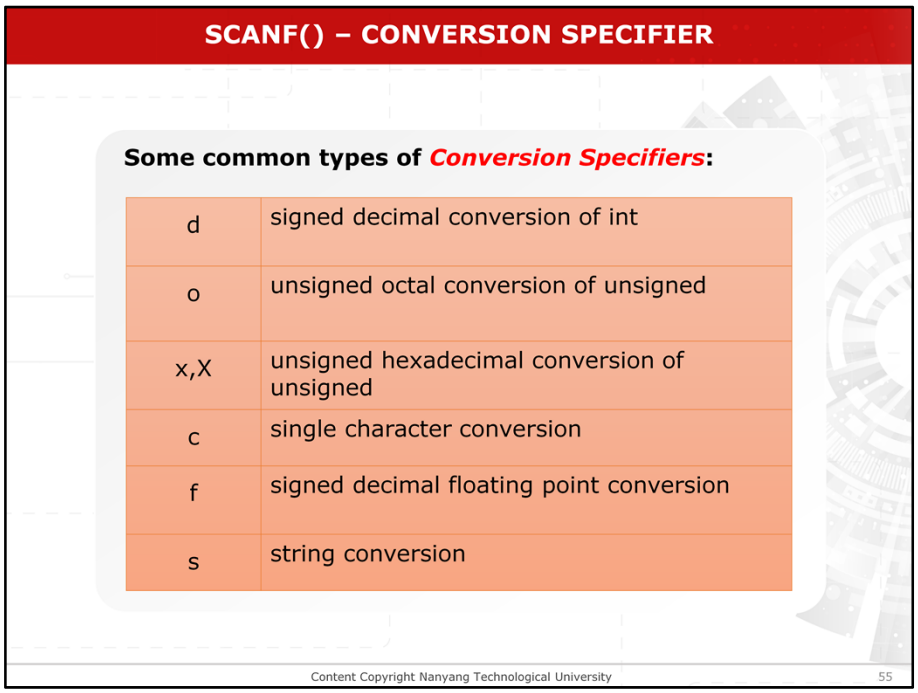


scanf() uses whitespace characters (tabs, spaces and newlines) to determine how to separate the input into different fields to be stored.



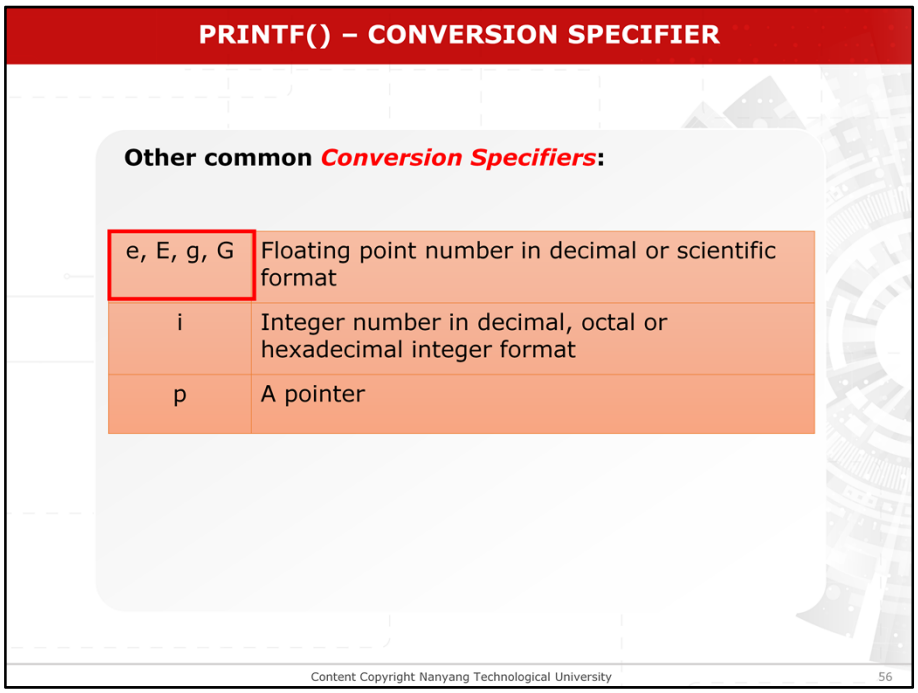
The options provide the additional control on how to print a value as follows:

Flag controls the display of plus or minus sign of a number, the display of the decimal point in floating point numbers, and left or right justification. A **flag** can be any of the five values: **+**, **-**, **space**, **#** and **0**. Zero or more flags may be present.



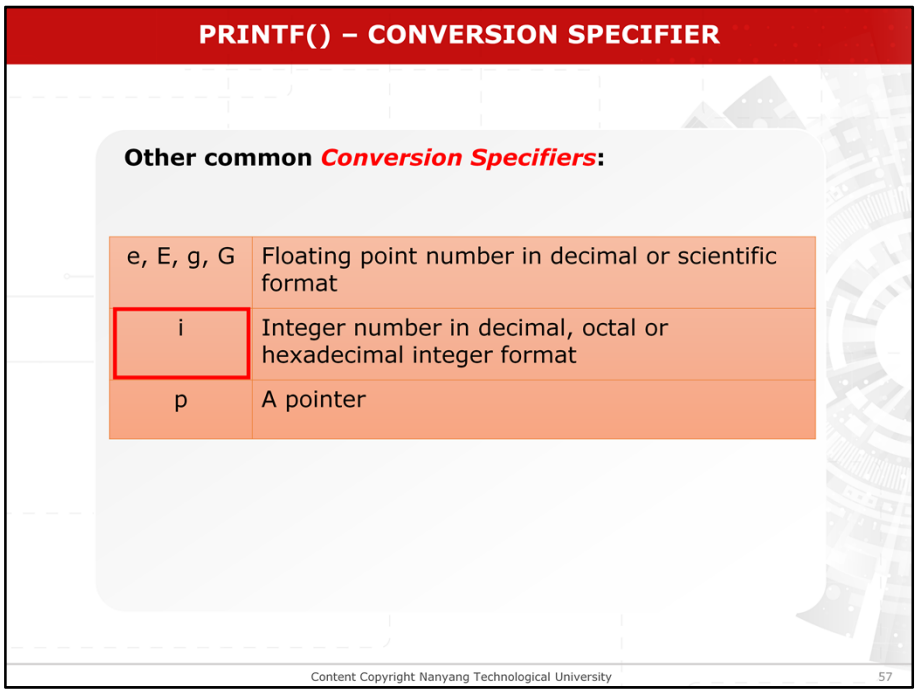
Conversion Specifier

The most common conversion specifiers are **d** for decimal integers, o for octal integers, x for hexadecimal conversion, **c** for characters, **f** for floating point numbers, and **s** for strings.



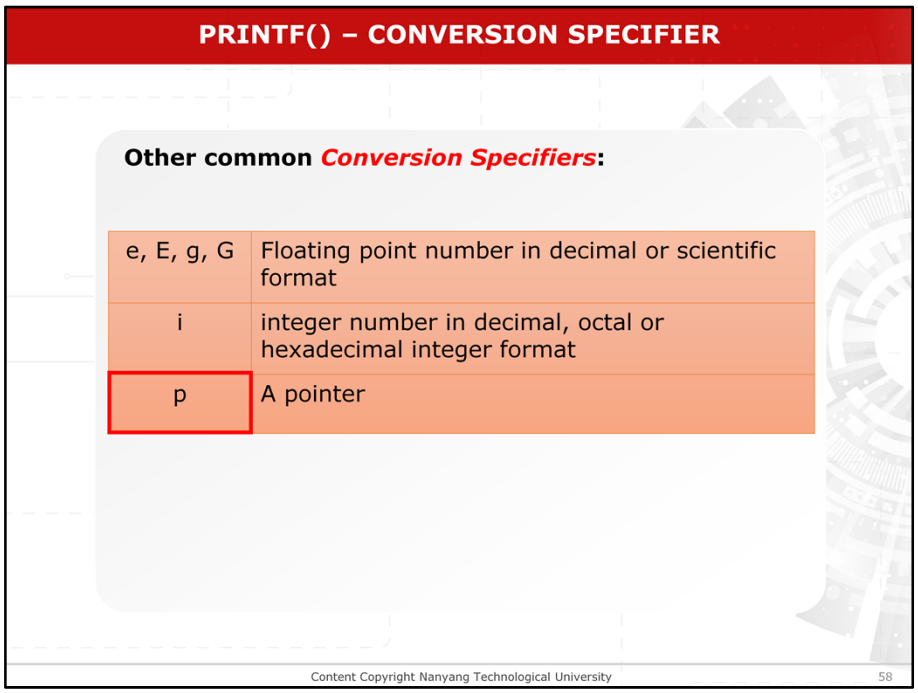
Other conversion specifiers include:

- **e, E, g, G** - floating point number in decimal or scientific format.



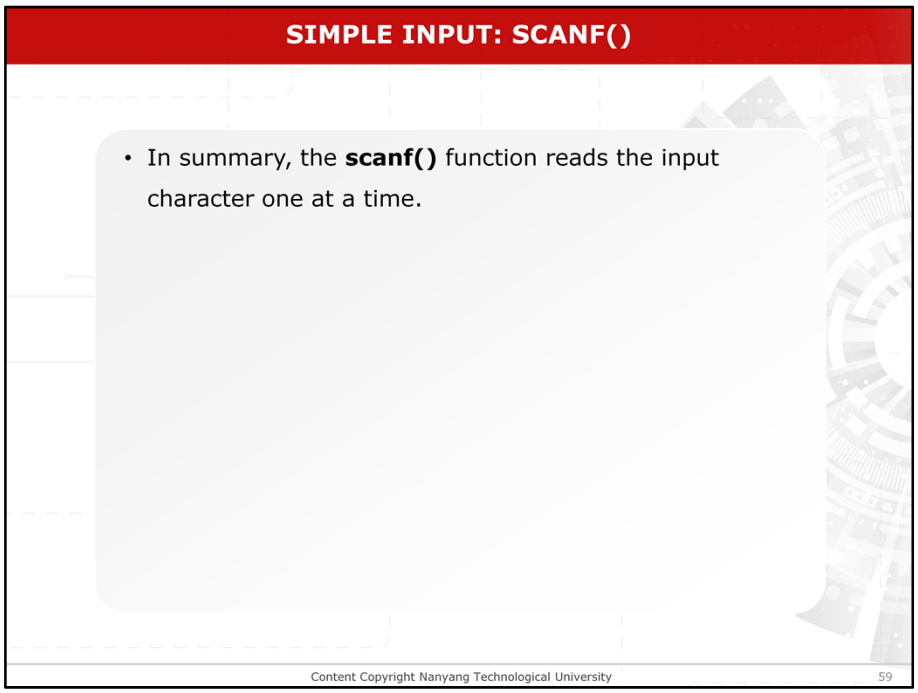
Other conversion specifiers include:

- **e, E, g, G** - floating point number in decimal or scientific format.

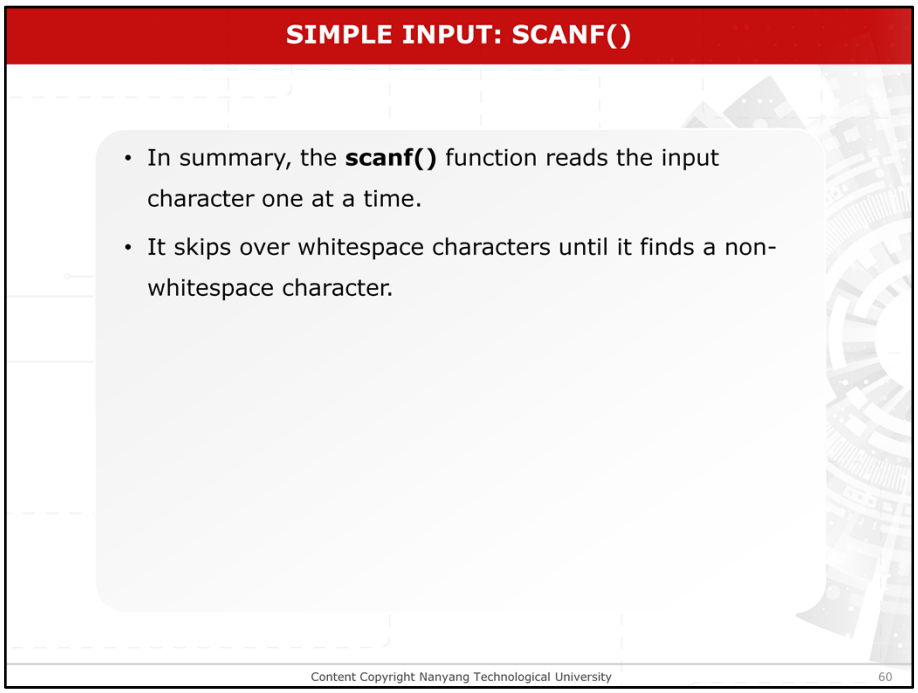


Other conversion specifiers include:

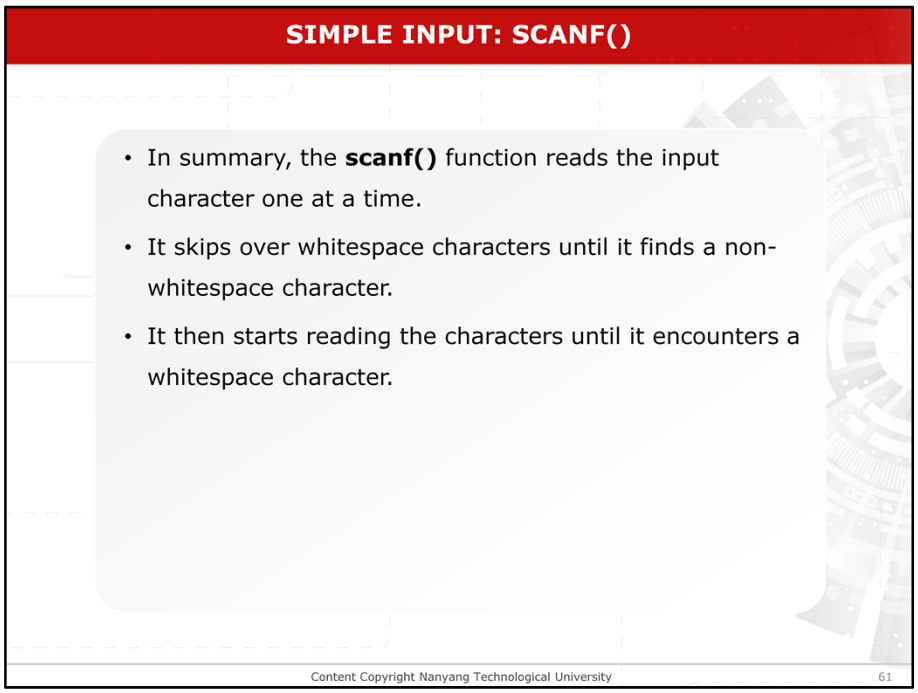
- **e, E, g, G** - floating point number in decimal or scientific format.



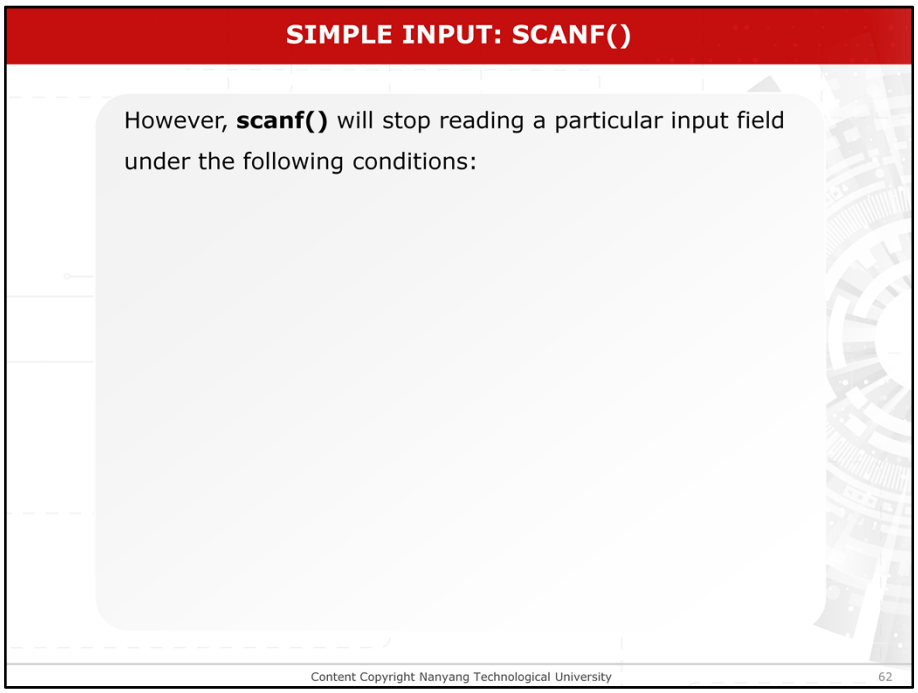
In summary, the **scanf()** function reads the input character one at a time.



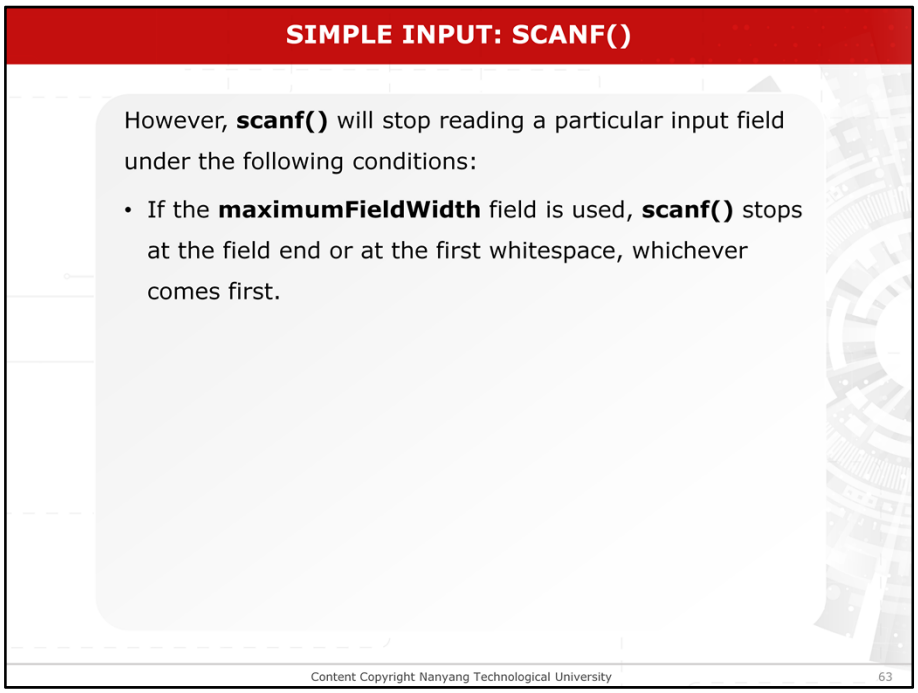
It skips over whitespace characters until it finds a non-whitespace character.



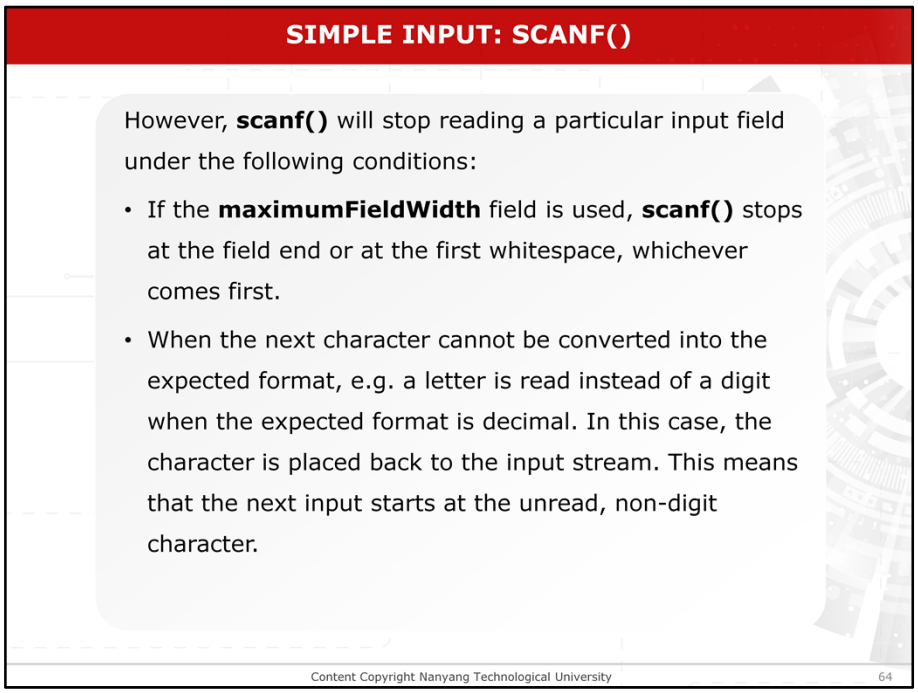
It skips over whitespace characters until it finds a non-whitespace character.



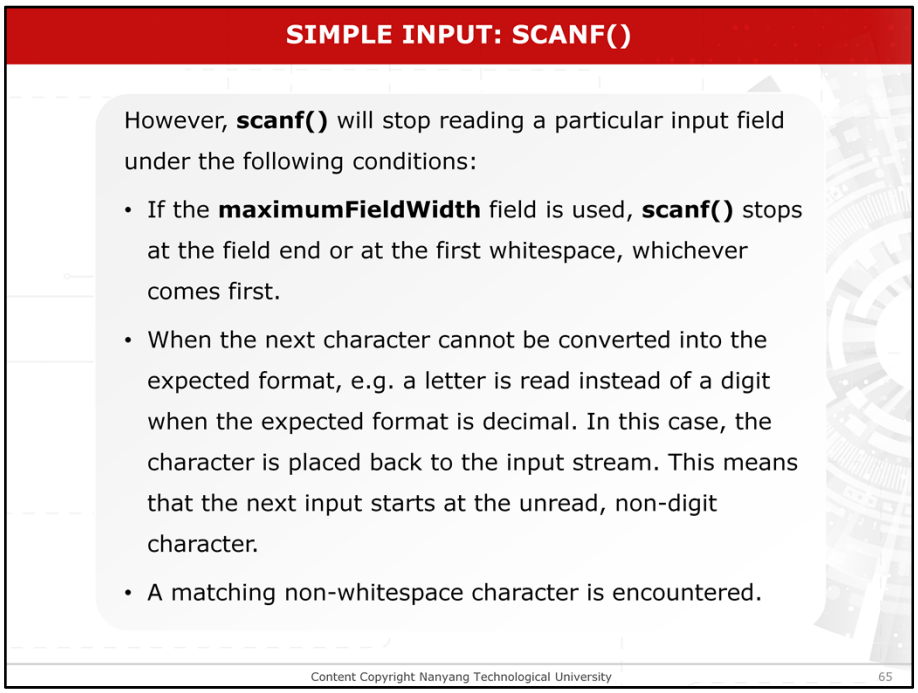
. However, **scanf()** will stop reading a particular input field under the following conditions:



- If the **maximumFieldWidth** field is used, **scanf()** stops at the field end or at the first whitespace, whichever comes first.



.When the next character cannot be converted into the expected format, e.g. a letter is read instead of a digit when the expected format is decimal. In this case, the character is placed back to the input stream. This means that the next input starts at the unread, non-digit character.



- A matching non-whitespace character is encountered.

SCANF(): EXAMPLE

- A scanf() statement has the form:
  - **scanf (control-string, argument-list);**

```
#include <stdio.h>
int main( )
{
     int n1, n2;
     float f1;
     double f2;

     printf("Please enter 2 integers:\n");
     scanf("%d %d", &n1, &n2);
     printf("The sum = %d\n", n1+n2);
     printf("Please enter 2 floats:\n");
     scanf("%f %lf", &f1, &f2);
     // Note: use %lf for double data
     printf("The sum = %f\n", f1+f2);
     return 0;
}
```


The **scanf()** function is an input function that can be used to read formatted data. The **scanf()** function reads input character strings from the input device (e.g. keyboard), converts the strings character-by-character to values according to the specified format and then stores the values into the variables.

SCANF(): EXAMPLE

- A scanf() statement has the form:
  - **scanf (control-string, argument-list);**

```
#include <stdio.h>
int main( )
{
     int n1, n2;
     float f1;
     double f2;

     printf("Please enter 2 integers:\n");
     scanf("%d %d", &n1, &n2);
     printf("The sum = %d\n", n1+n2);
     printf("Please enter 2 floats:\n");
     scanf("%f %lf", &f1, &f2);
     // Note: use %lf for double data
     printf("The sum = %f\n", f1+f2);
     return 0;
}
```


In the program, the **scanf()** statement uses whitespace to separate user input.

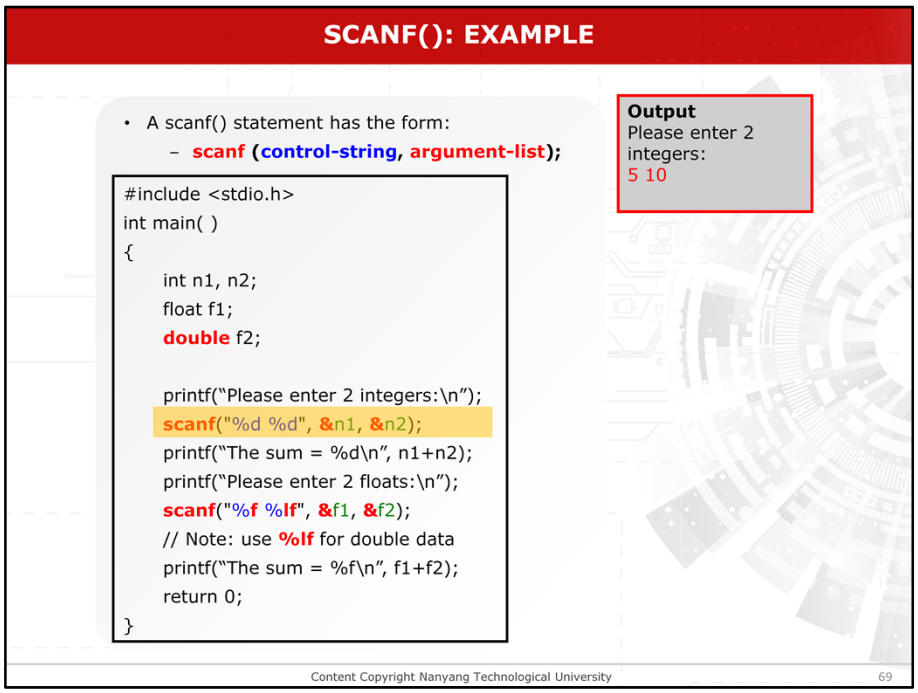
SCANF(): EXAMPLE

- A scanf() statement has the form:
  - **scanf (control-string, argument-list);**

```
#include <stdio.h>
int main( )
{
    int n1, n2;
    float f1;
    double f2;

    printf("Please enter 2 integers:\n");
    scanf("%d %d", &n1, &n2);
    printf("The sum = %d\n", n1+n2);
    printf("Please enter 2 floats:\n");
    scanf("%f %lf", &f1, &f2);
    // Note: use %lf for double data
    printf("The sum = %f\n", f1+f2);
    return 0;
}
```

Output
Please enter 2 integers:

No VO



User input is read by the scanf statement and it converts the strings character-by-character to values according to the specified format and then stores the values into the variables.

SCANF(): EXAMPLE

- A scanf() statement has the form:
  - **scanf (control-string, argument-list);**

```
#include <stdio.h>
int main( )
{
    int n1, n2;
    float f1;
    double f2;

    printf("Please enter 2 integers:\n");
    scanf("%d %d", &n1, &n2);
    printf("The sum = %d\n", n1+n2);
    printf("Please enter 2 floats:\n");
    scanf("%f %lf", &f1, &f2);
    // Note: use %lf for double data
    printf("The sum = %f\n", f1+f2);
    return 0;
}
```

Output
Please enter 2 integers:
5 10
The sum = 15

THE SUM IS PRINTED ON THE SCREEN

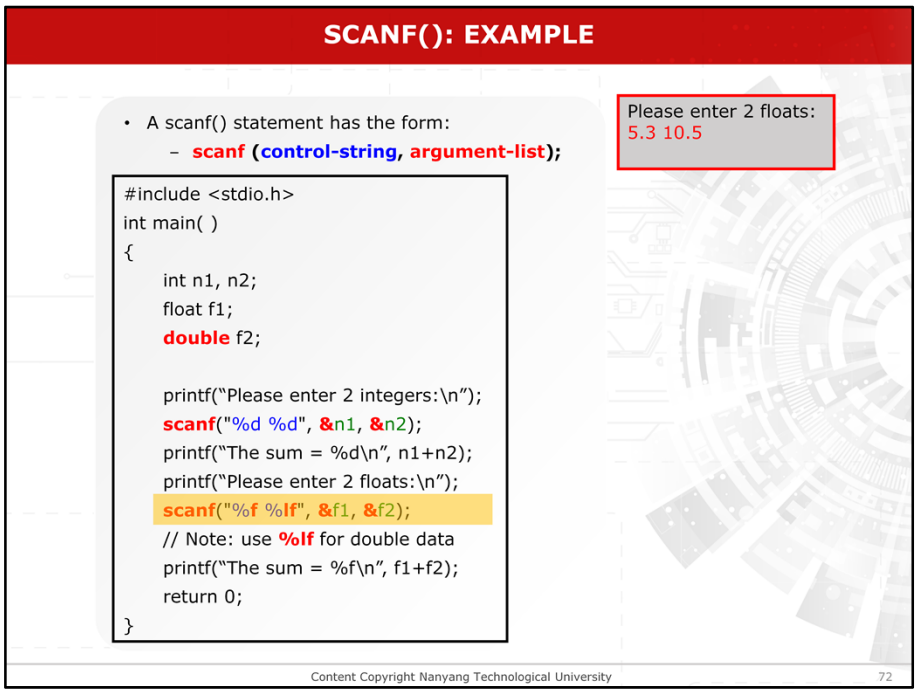
SCANF(): EXAMPLE

- A scanf() statement has the form:
  - **scanf (control-string, argument-list);**

```
#include <stdio.h>
int main( )
{
    int n1, n2;
    float f1;
    double f2;

    printf("Please enter 2 integers:\n");
    scanf("%d %d", &n1, &n2);
    printf("The sum = %d\n", n1+n2);
    printf("Please enter 2 floats:\n");
    scanf("%f %lf", &f1, &f2);
    // Note: use %lf for double data
    printf("The sum = %f\n", f1+f2);
    return 0;
}
```

```
Please enter 2 floats:
```


NO VO



The **sizeSpecification "%lf"** is used to read a value in **double** data type. The **scanf()** function can also return the number of items that it has successfully read.

SCANF(): EXAMPLE

- A scanf() statement has the form:
  - **scanf (control-string, argument-list);**

```
#include <stdio.h>
int main( )
{
    int n1, n2;
    float f1;
    double f2;

    printf("Please enter 2 integers:\n");
    scanf("%d %d", &n1, &n2);
    printf("The sum = %d\n", n1+n2);
    printf("Please enter 2 floats:\n");
    scanf("%f %lf", &f1, &f2);
    // Note: use %lf for double data
    printf("The sum = %f\n", f1+f2);
    return 0;
}
```

```
Please enter 2 floats:
5.3 10.5
The sum = 15.800000
```

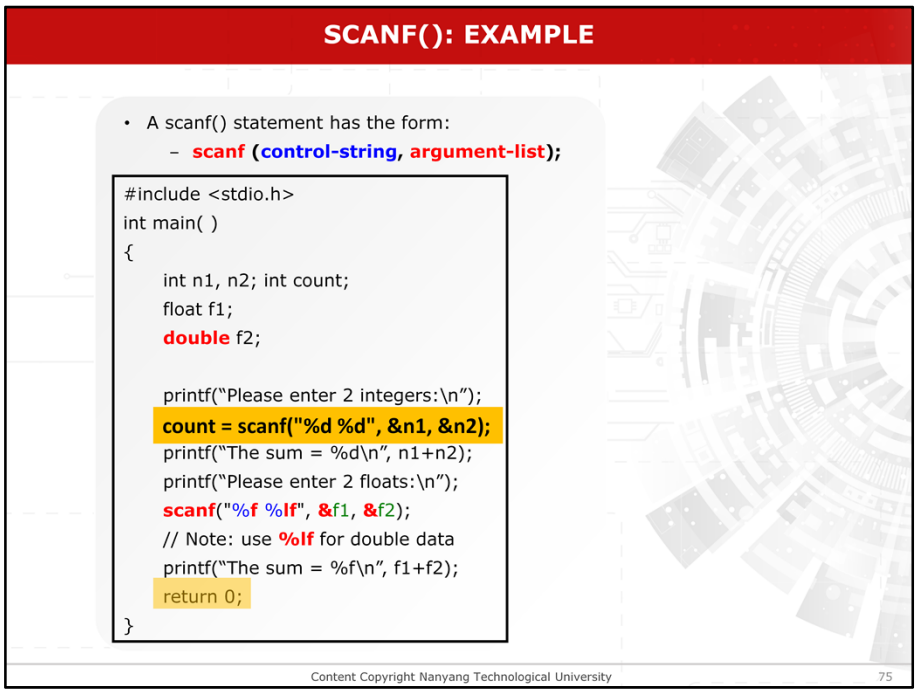

THE SUM IS PRINTED ON THE SCREEN

SCANF(): EXAMPLE

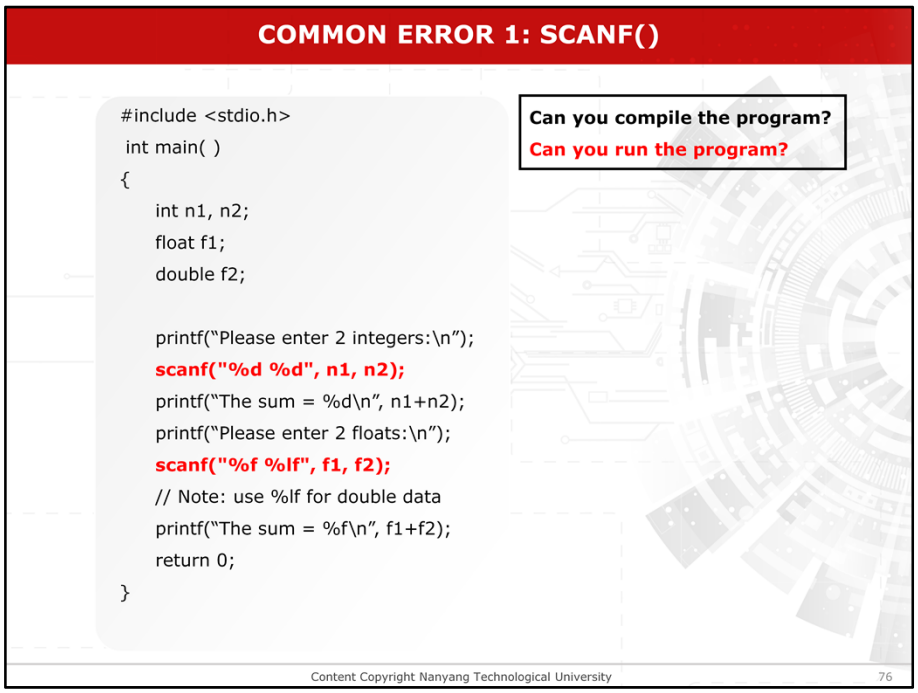
- A scanf() statement has the form:
  - **scanf (control-string, argument-list);**

```
#include <stdio.h>
int main( )
{
     int n1, n2;
     float f1;
     double f2;

     printf("Please enter 2 integers:\n");
     scanf("%d %d", &n1, &n2);
     printf("The sum = %d\n", n1+n2);
     printf("Please enter 2 floats:\n");
     scanf("%f %lf", &f1, &f2);
     // Note: use %lf for double data
     printf("The sum = %f\n", f1+f2);
     return 0;
}
```


If no item is read, **scan f()** returns the value 0. If end-of-file is encountered, it returns **E O F**.



We can modify the first **scanf()** statement in the program to return the number of items read as the one shown here, where **count** is a variable of data type **int**.



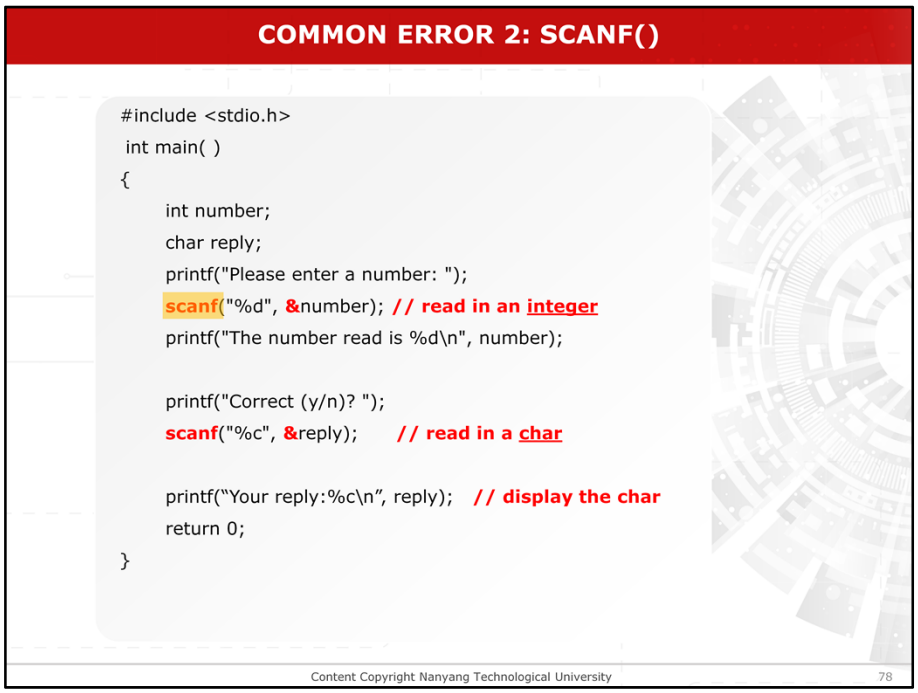
One of the most common errors made by novice programmers is the omission of the address operator preceded before the specified variable when writing the scan f() function.

COMMON ERROR 1: SCANF()

```
#include <stdio.h>
int main( )
{
    int n1, n2;
    float f1;
    double f2;

    printf("Please enter 2 integers:\n");
    scanf("%d %d", &n1, &n2);
    printf("The sum = %d\n", n1+n2);
    printf("Please enter 2 floats:\n");
    scanf("%f %lf", &f1, &f2);
    // Note: use %lf for double data
    printf("The sum = %f\n", f1+f2);
    return 0;
}
```

Can you compile the program?
Can you run the program?

In such cases, the program will be able to compile, but unable to run correctly. The program will be terminated abruptly.



Common Error 2

In the program, it implements the input operation using the **scanf()** statement.

COMMON ERROR 2: SCANF()

```
#include <stdio.h>
int main( )
{
    int number;
    char reply;
    printf("Please enter a number: ");
    scanf("%d", &number); // read in an integer
    printf("The number read is %d\n", number);

    printf("Correct (y/n)? ");
    scanf("%c", &reply);     // read in a char

    printf("Your reply:%c\n", reply);   // display the char
    return 0;
}
```

Intended Output:
Please enter a number:

 79

The intended execution of the program is that it first prompts the user to enter a number

COMMON ERROR 2: SCANF()

```
#include <stdio.h>
 int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("Your reply:%c\n", reply);   // display the char
      return 0;
}
```

Intended Output:
Please enter a number:
1234<Enter>

then reads in the number with the **scanf()** function

COMMON ERROR 2: SCANF()

```
#include <stdio.h>
 int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("Your reply:%c\n", reply);   // display the char
      return 0;
}
```

Intended Output:
Please enter a number:
1234<Enter>
The number read is 1234

NO VO

COMMON ERROR 2: SCANF()

```
#include <stdio.h>
 int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("Your reply:%c\n", reply);   // display the char
      return 0;
}
```

Intended Output:

Please enter a number:
1234<Enter>
The number read is 1234
Correct (y/n)? y

and asks the user whether the number is correct.

COMMON ERROR 2: SCANF()

```
#include <stdio.h>
int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("Your reply:%c\n", reply);   // display the char
      return 0;
}
```

Intended Output:
Please enter a number:
1234<Enter>
The number read is 1234
Correct (y/n)? y

The reply will be read in as a single character, yes or no using **scanf()**, and will be stored in the variable **reply**

COMMON ERROR 2: SCANF()

```
#include <stdio.h>
 int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("Your reply:%c\n", reply);  // display the char
      return 0;
}
```

Intended Output:

Please enter a number:
1234<Enter>
The number read is 1234
Correct (y/n)? **y**
Your reply : y

Then, the user's reply will be printed to the screen.

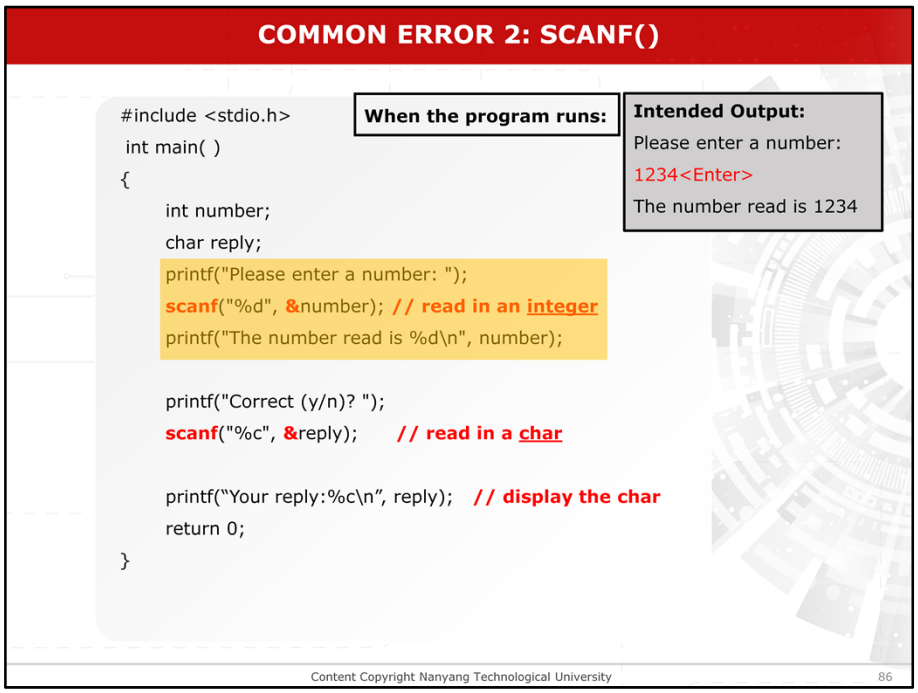
COMMON ERROR 2: SCANF()

```
#include <stdio.h>
 int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("Your reply:%c\n", reply);   // display the char
      return 0;
}
```

Intended Output:
Please enter a number:
1234<Enter>
The number read is 1234
Correct (y/n)? **y**
Your reply : y

Can you compile the program?
Can you run the program as intended?

Is there any syntax error in this program? If no, can the program run as intended



When the program is written and run, after the number has been entered with the **Enter** key, the number is read.

COMMON ERROR 2: SCANF()

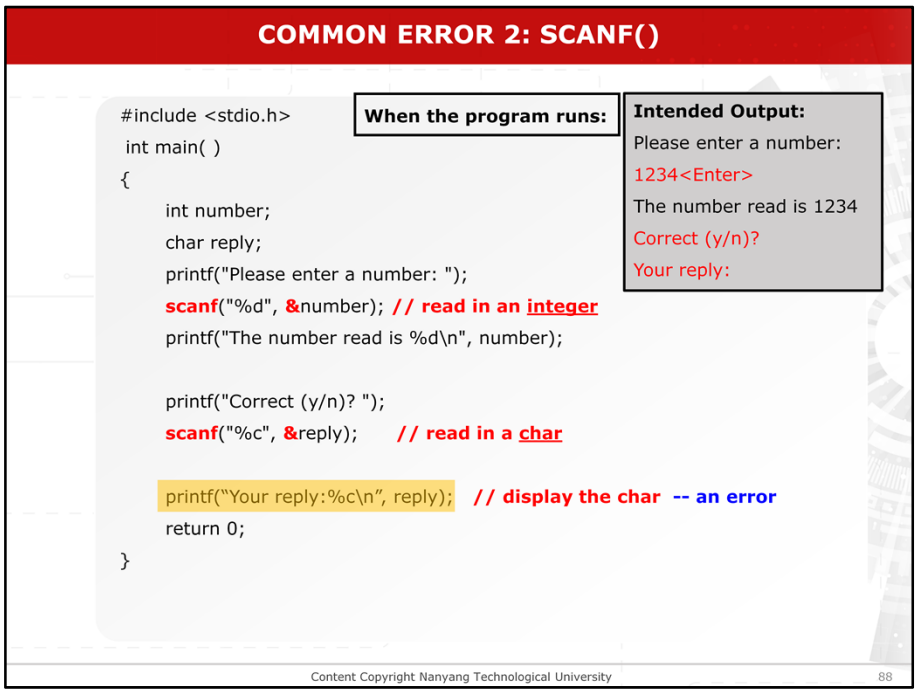
```
#include <stdio.h>
 int main( )
{
      int number;
      char reply;
      printf("Please enter a number: ");
      scanf("%d", &number); // read in an integer
      printf("The number read is %d\n", number);

      printf("Correct (y/n)? ");
      scanf("%c", &reply);      // read in a char

      printf("Your reply:%c\n", reply);   // display the char  -- an error
      return 0;
}
```

When the program runs:

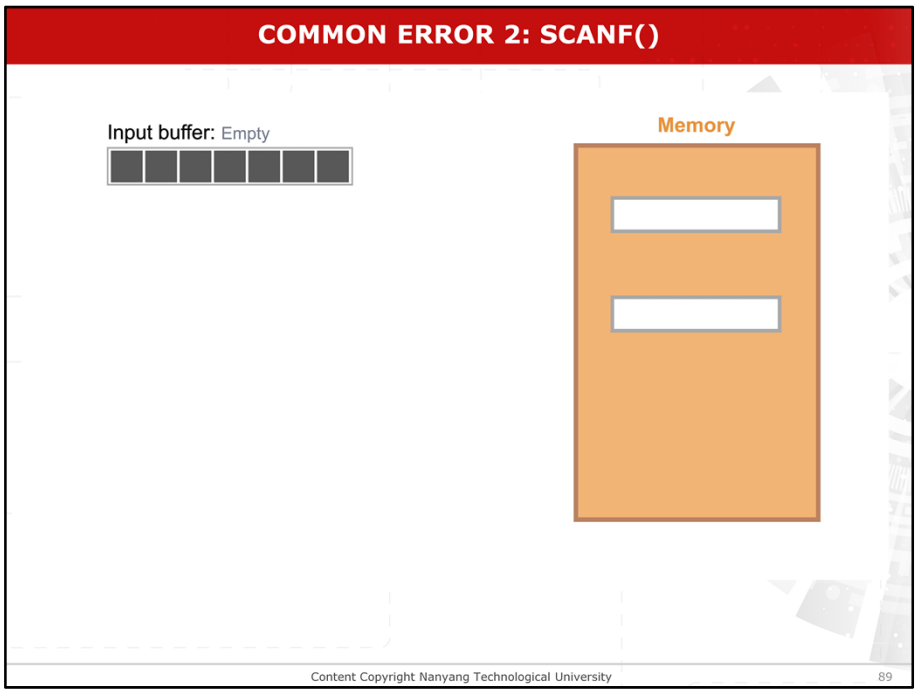
Intended Output:

Please enter a number:
1234<Enter>
The number read is 1234
Correct (y/n)?
Your reply:

When the program prompts the user with the message "**Correct (yes/no)?**", it does not wait for any new user input

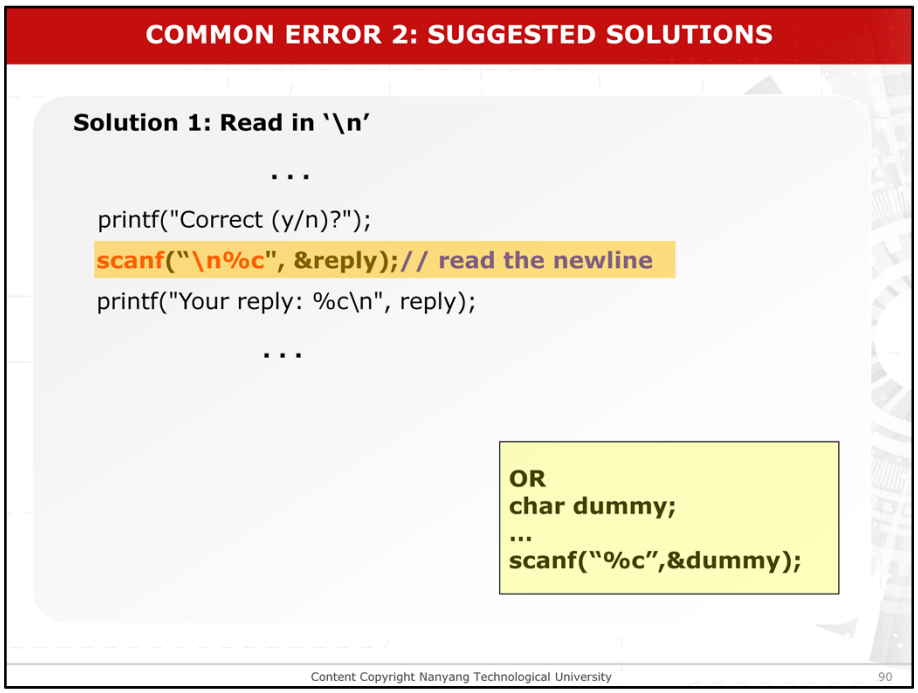


Instead, the new line character will be displayed as the reply by the user. As such, the program does not run correctly as intended. It means that the program does not read in user reply on **y** or **n**.

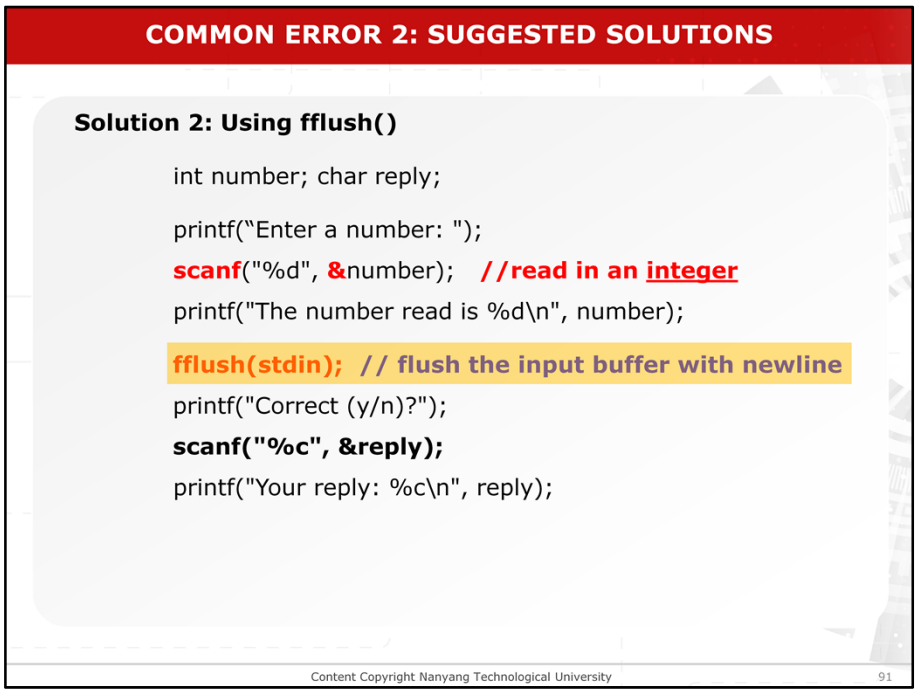


Common Error 2: Reason

When the user inputs 1234<Enter>, the value will first be stored in the input buffer. After executing **scanf**("%d", **address** number); the value 1234 will then be assigned to the variable **number**. When executing **scanf**("%c", **address** reply); the program tries to read in the user input from the input buffer. As the input buffer still stores the newline character '\n', which will then be used and assigned to the variable **reply**. The program does not wait for user to enter his reply.

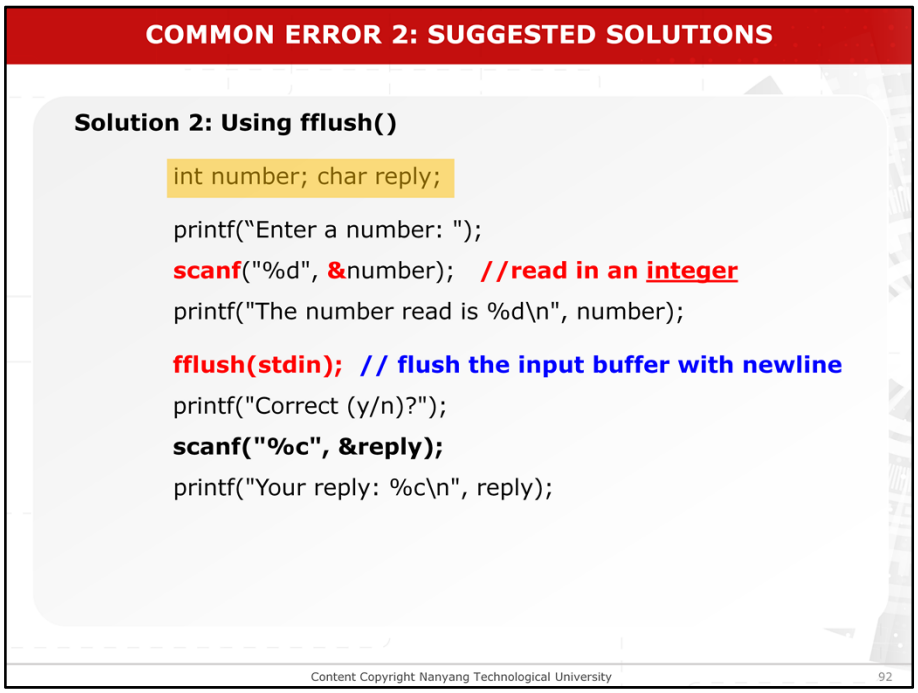


There are two ways to avoid this problem:
The first way is to modify the **scanf()** to read in a newline from the buffer.

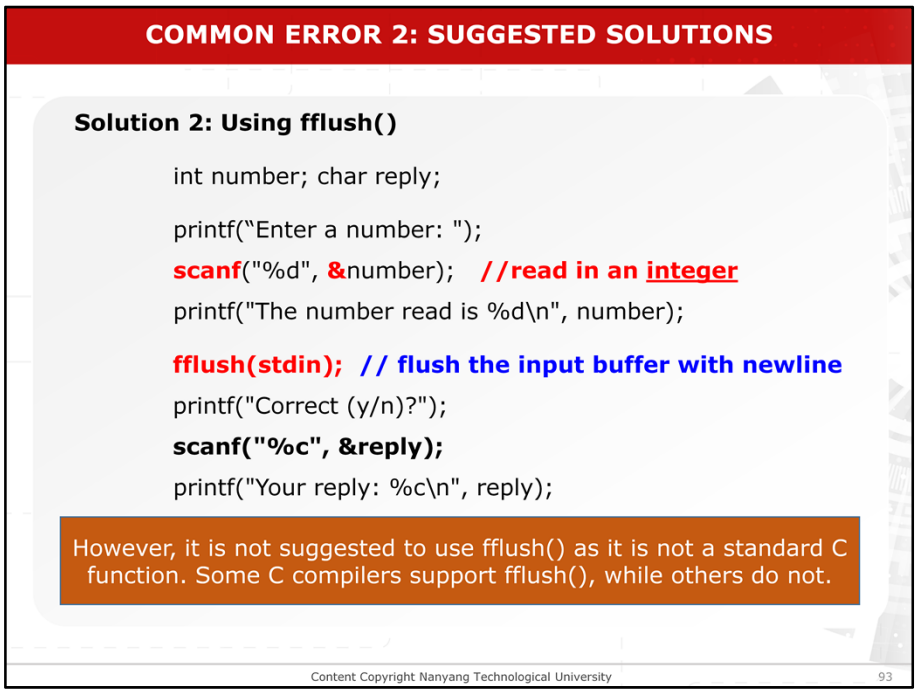


The second way is to place the **fflush(stdin)** function before **scanf()** as shown here;

As such, it will flush or empty the input buffer that contains the newline character before asking for reply. **fflush()** is a standard C library function.

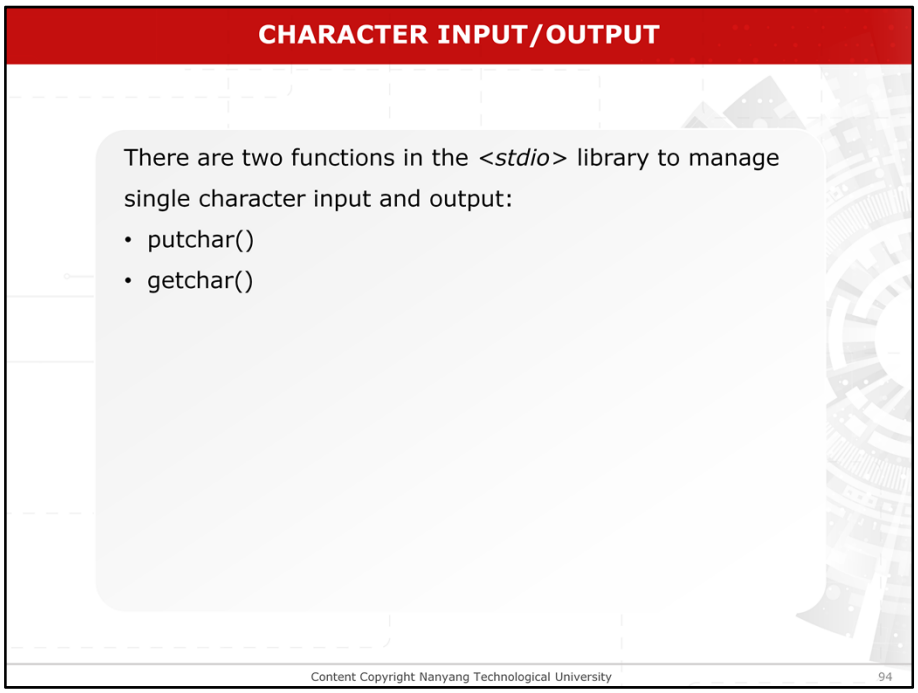


Note: Generally, this problem may occur when the programs read in numerical data and character data one after another. For example, in this example, the program first reads in an integer number, then followed by reading in a character. If the program reads in only integer numbers, such problem will not occur.

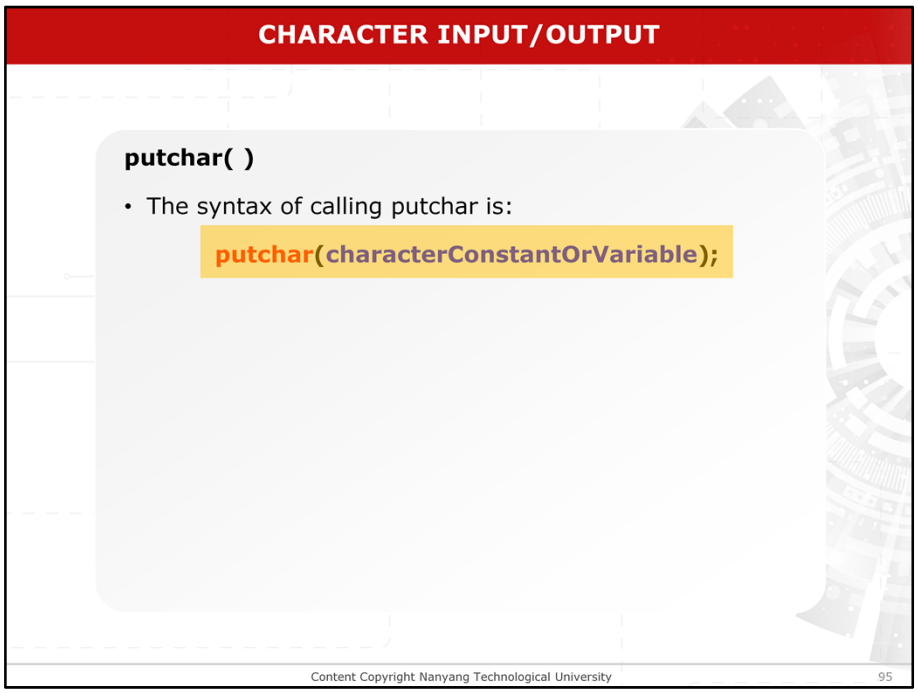


The second way is to place the **fflush(stdin)** function before **scanf()** as shown here;

As such, it will flush or empty the input buffer that contains the newline character before asking for reply.



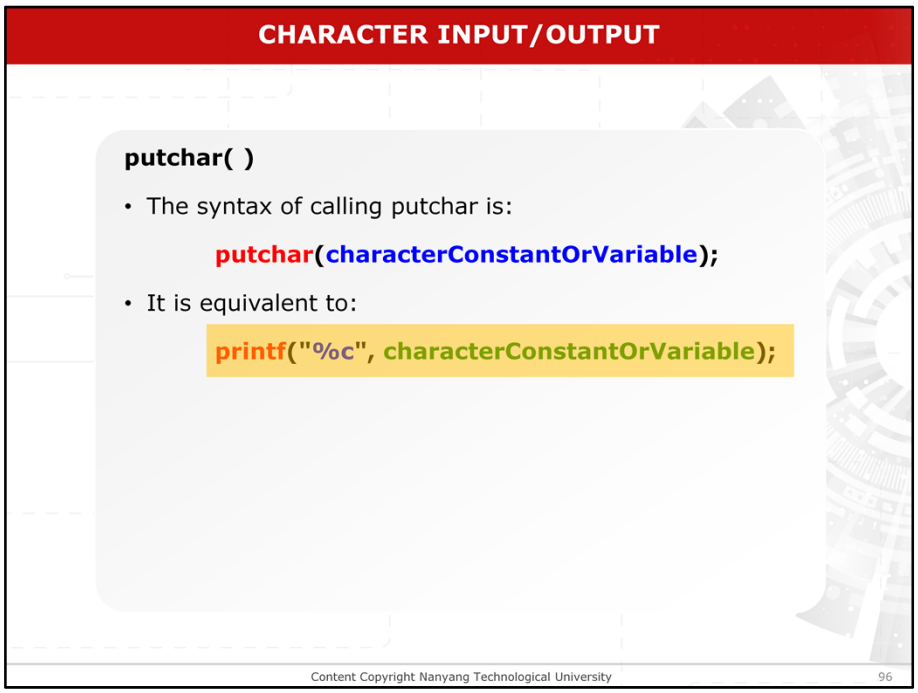
There are two functions in the *<stdio>* library to manage single character input and output: **putchar()** and **getchar()**.



putchar()

The function **putchar()** takes a single argument, and prints the character. The syntax of calling **putchar()** is

shown here



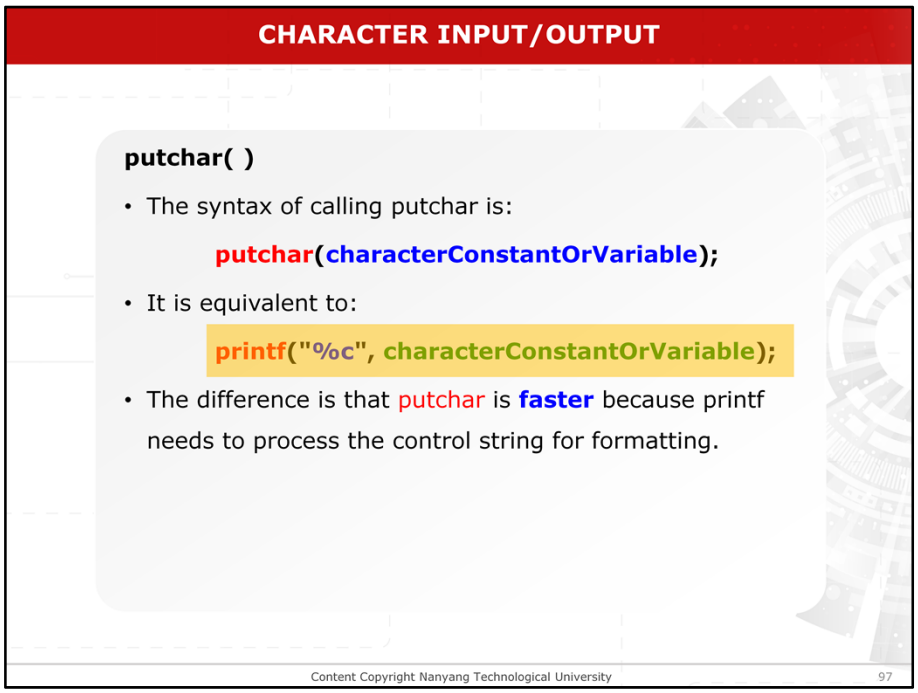
putchar()

The function **putchar()** takes a single argument, and prints the character. The syntax of calling **putchar()** is

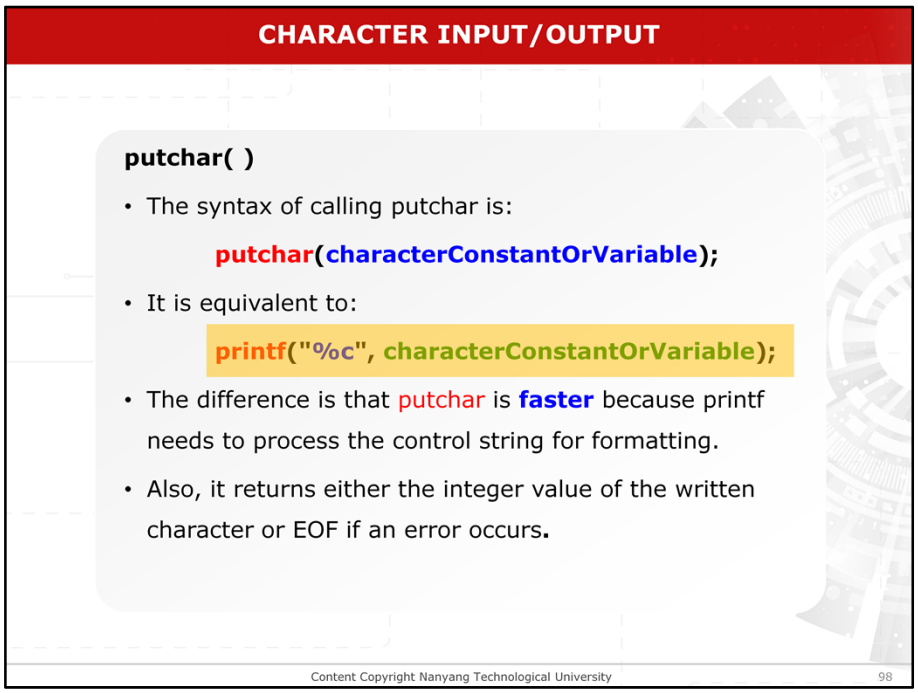
shown here

which is equivalent to

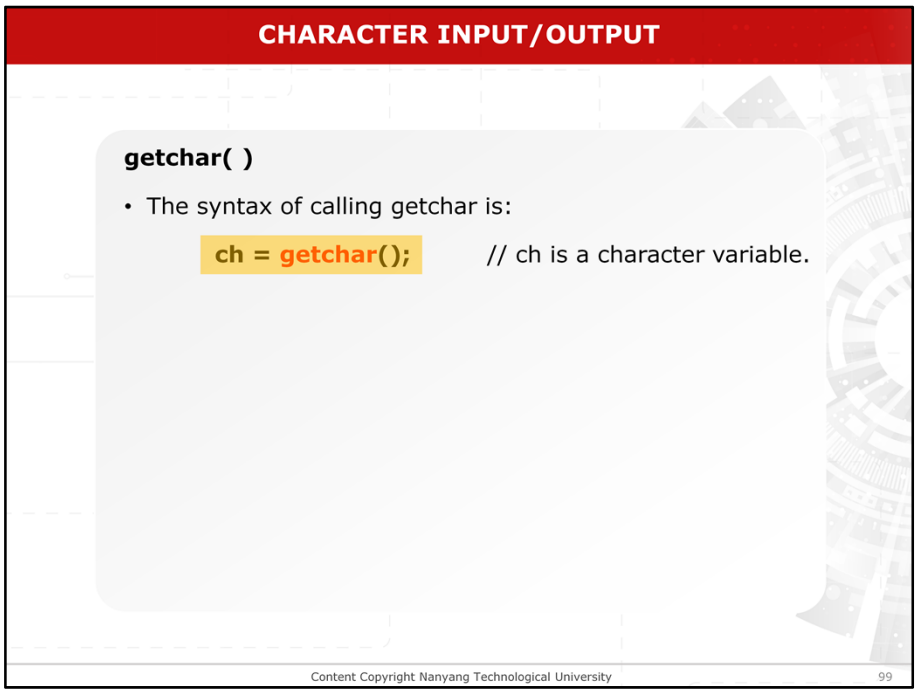
printf("%c", characterConstantOrVariable);



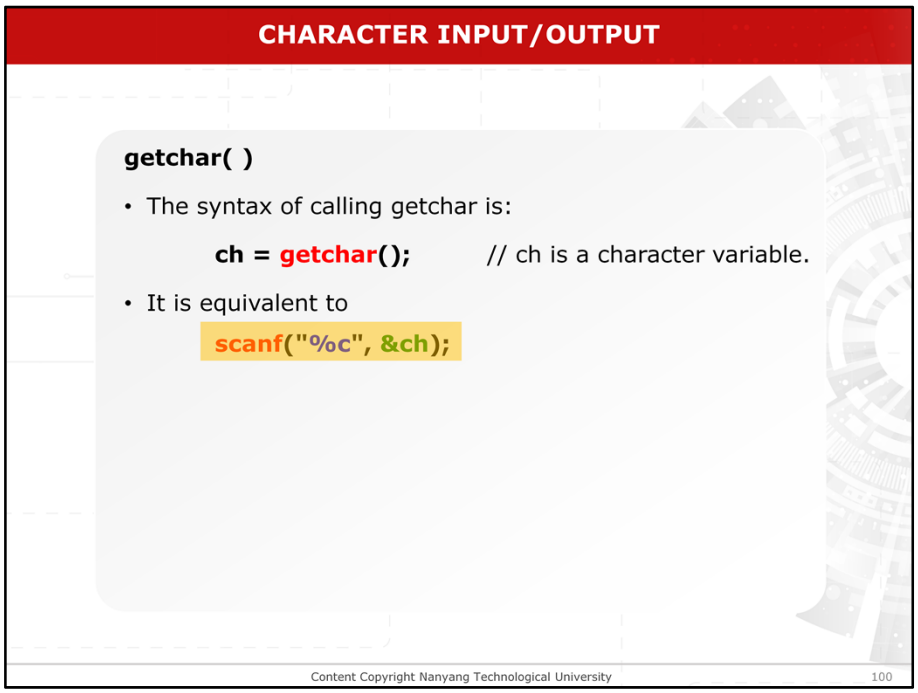
The difference is that **putchar()** is faster because **printf()** needs to process the control-string for formatting.



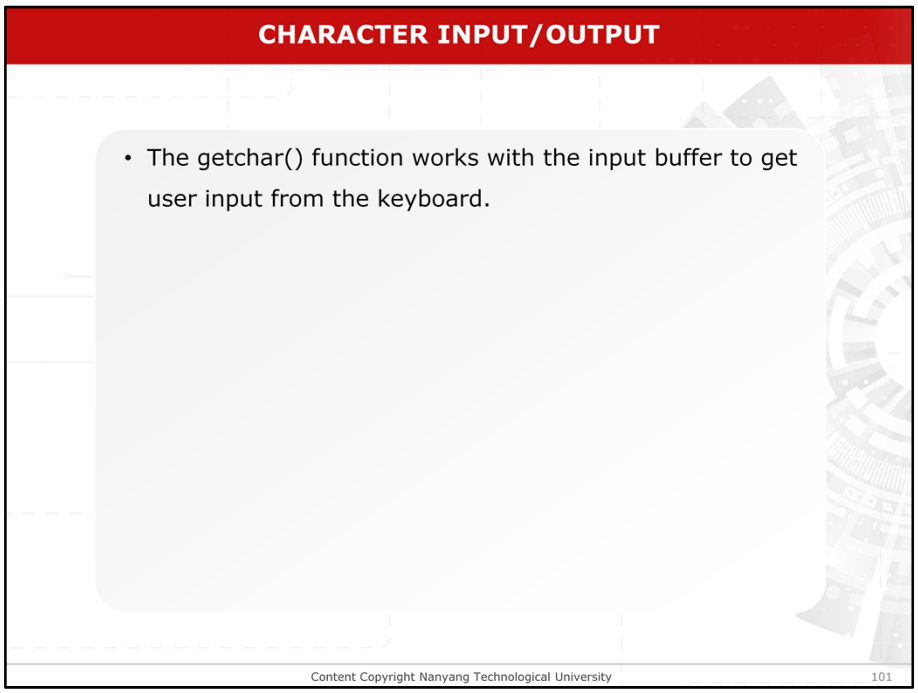
Also, it returns either the integer value of the written character or EOF if an error occurs**.**



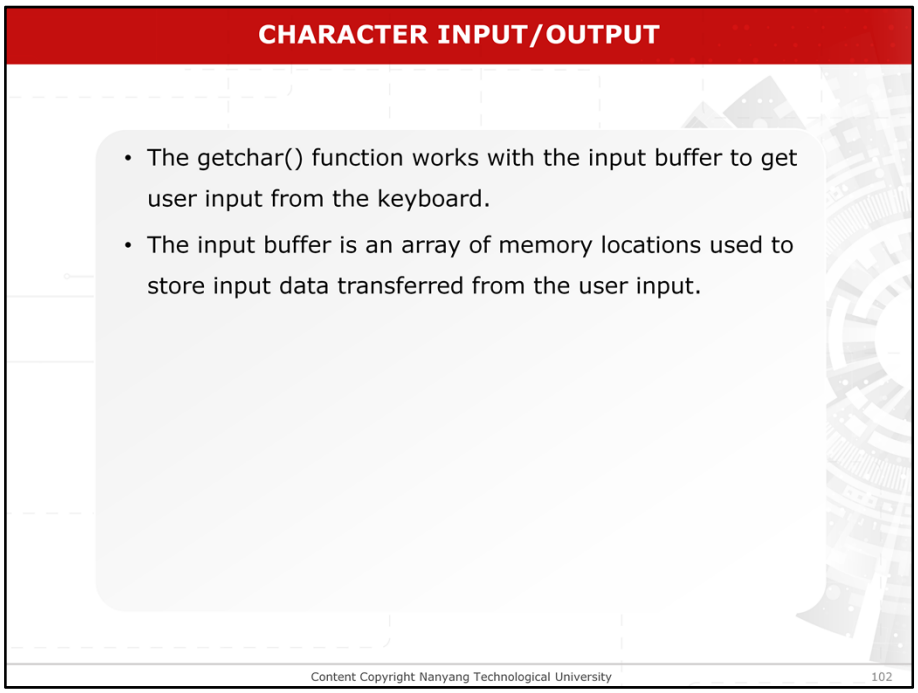
THE SYNTAX OF CALLING getchar() IS SHOWN HERE, where **ch** is a character variable. The **getchar()** function returns the next character from the input, or **EOF** if end-of-file is reached or if an error occurs. No arguments are required for **getchar()**.



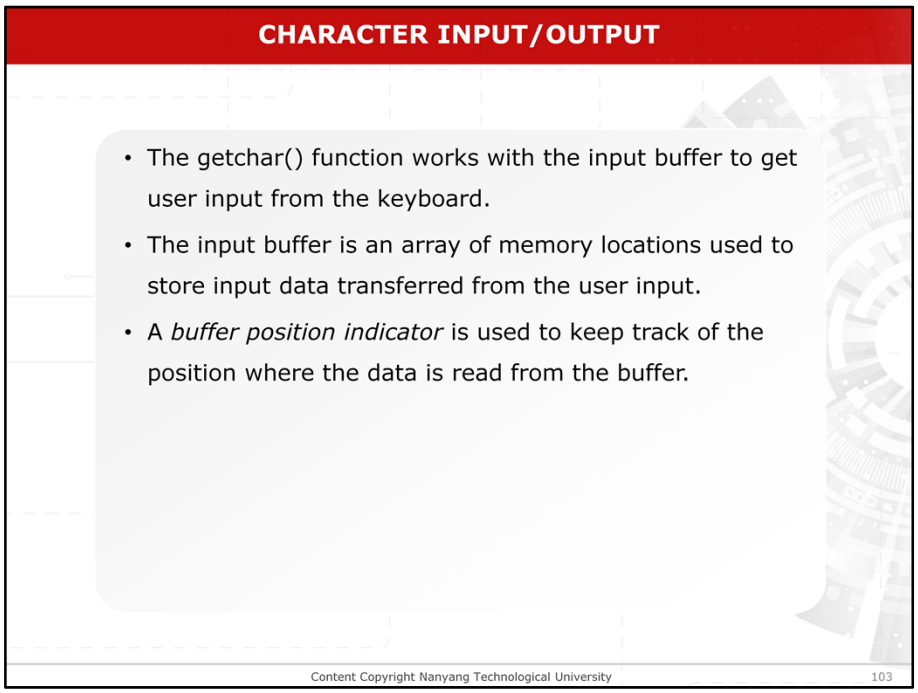
It is equivalent to the statement shown here.



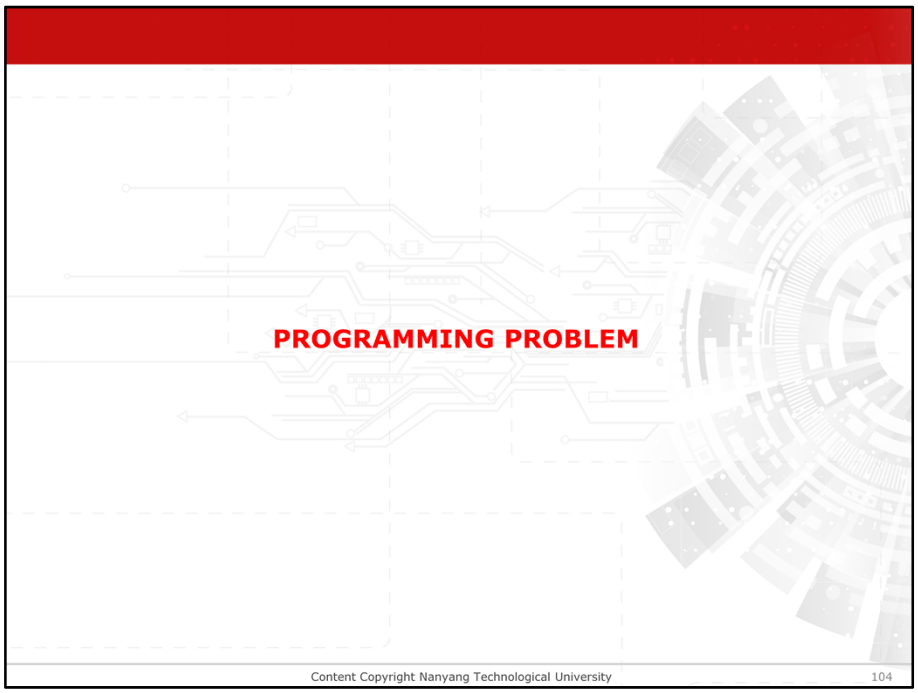
The **getchar()** function works with the input buffer to get user input from the keyboard.

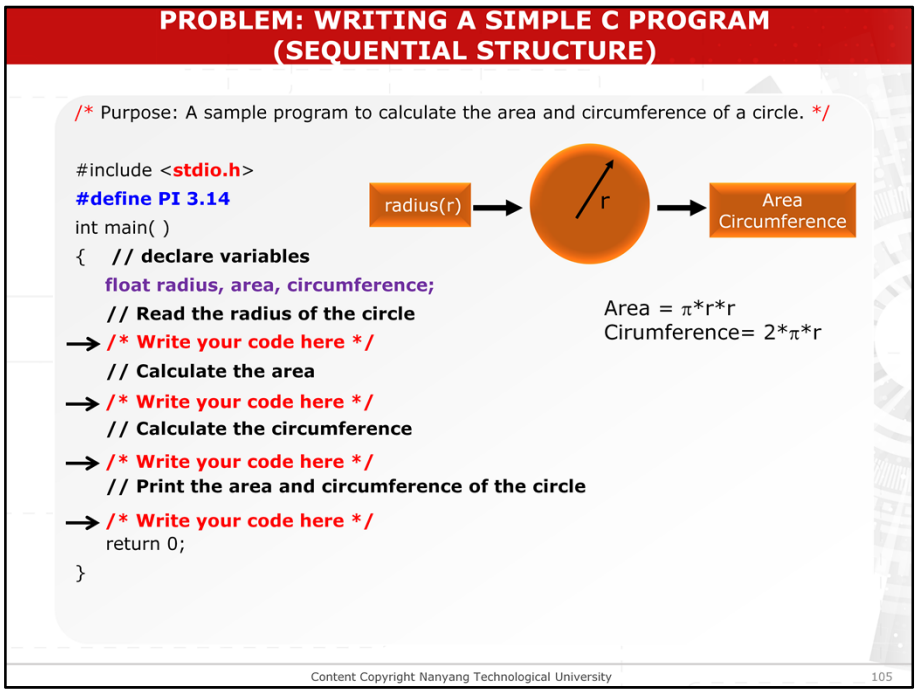


The input buffer is an array of memory locations used to store input data transferred from the user input.



A *buffer position indicator* is used to keep track of the position where the data is read from the buffer.


PROGRAMMING PROBLEM
Content Copyright Nanyang Technological University
104

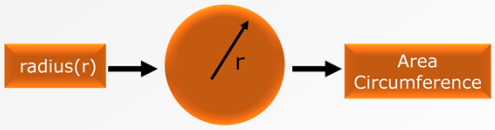


Programming Problem

In this problem, write a program that computes the area and circumference of a circle. The **printf()** and **scanf()** functions are used for I/O operations. The assignment statements are used in programming the logic.

PROBLEM: WRITING A SIMPLE C PROGRAM (SEQUENTIAL STRUCTURE)

```
/* Purpose: A sample program to calculate the area and circumference of a circle. */

#include <stdio.h>
#define PI 3.14
int main( )
{   // declare variables
    float radius, area, circumference;
    // Read the radius of the circle
    printf(Enter the radius:");
    scanf("%",&radius);
    // Calculate the area
    area = PI * radius * radius;
    // Calculate the circumference
    circumference = 2 * PI * radius;
    // Print the area and circumference of the circle
    printf("The area is %f\n", area);
    printf("The circumference is %f", circumference );
    return 0;
}
```



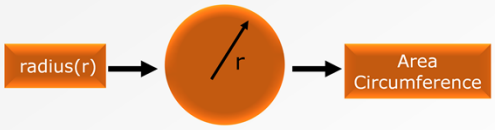
Area = π*r*r
Cirumference= 2*π*r

Output
Enter the radius: 5.0

The programs reads in the radius of the circle.

PROBLEM: WRITING A SIMPLE C PROGRAM (SEQUENTIAL STRUCTURE)

```
/* Purpose: A sample program to calculate the area and circumference of a circle. */

#include <stdio.h>
#define PI 3.14
int main( )
{   // declare variables
    float radius, area, circumference;
    // Read the radius of the circle
    printf(Enter the radius:");
    scanf("%",&radius);
    // Calculate the area
    area = PI * radius * radius;
    // Calculate the circumference
    circumference = 2 * PI * radius;
    // Print the area and circumference of the circle
    printf("The area is %f\n", area);
    printf("The circumference is %f", circumference );
    return 0;
}
```



Area = π*r*r
Cirumference= 2*π*r

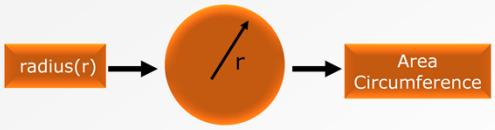
Output
Enter the radius: 5.0

 107

computes the area and circumference,
.

PROBLEM: WRITING A SIMPLE C PROGRAM (SEQUENTIAL STRUCTURE)

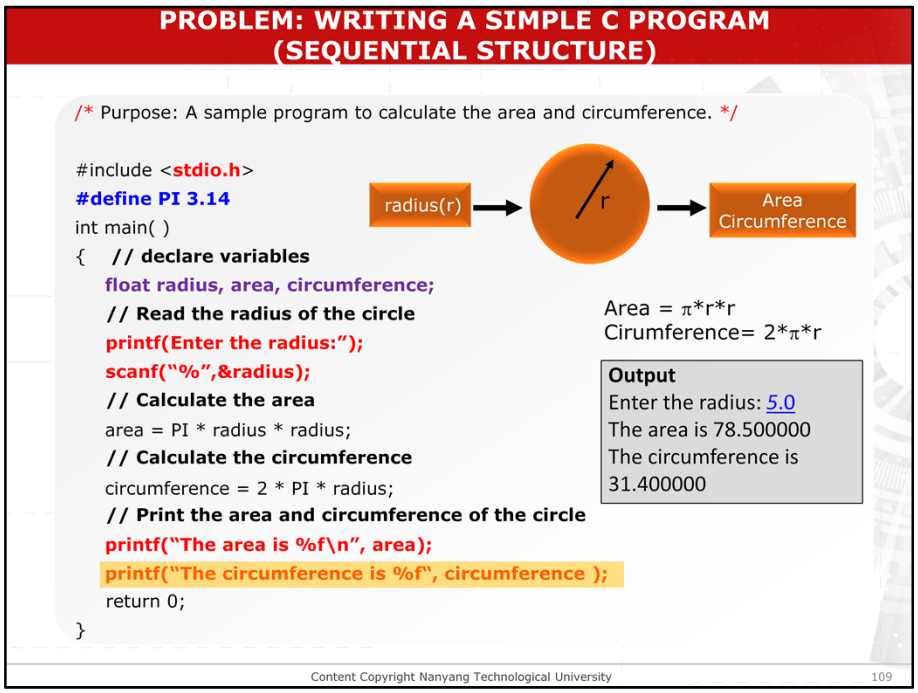
```
/* Purpose: A sample program to calculate the area and circumference. */

#include <stdio.h>
#define PI 3.14
int main( )
{   // declare variables
    float radius, area, circumference;
    // Read the radius of the circle
    printf(Enter the radius:");
    scanf("%",&radius);
    // Calculate the area
    area = PI * radius * radius;
    // Calculate the circumference
    circumference = 2 * PI * radius;
    // Print the area and circumference of the circle
    printf("The area is %f\n", area);
    printf("The circumference is %f", circumference );
    return 0;
}
```



Area = π*r*r
Cirumference= 2*π*r

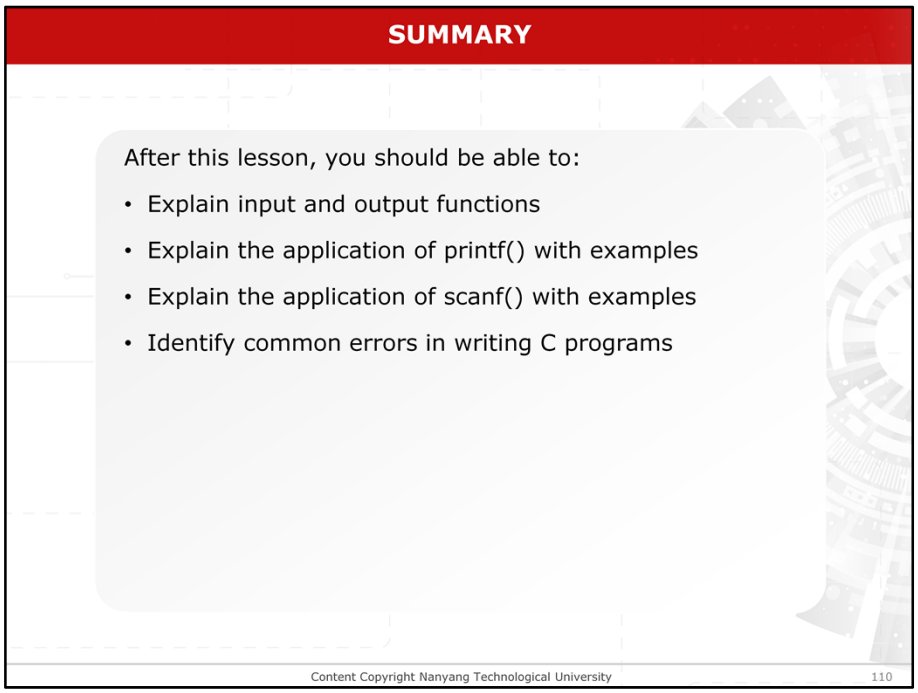
Output
Enter the radius: 5.0
The area is 78.500000

and displays the area and circumference on the screen.



Programming Problem

In this problem, write a program that computes the area and circumference of a circle. The **printf()** and **scanf()** functions are used for I/O operations. The assignment statements are used in programming the logic.



After watching this video lesson, you should be able to do the points listed.