

This lesson is on Basic C-Programming

OVERVIEW

The following are the coverage for Basic C Programming:

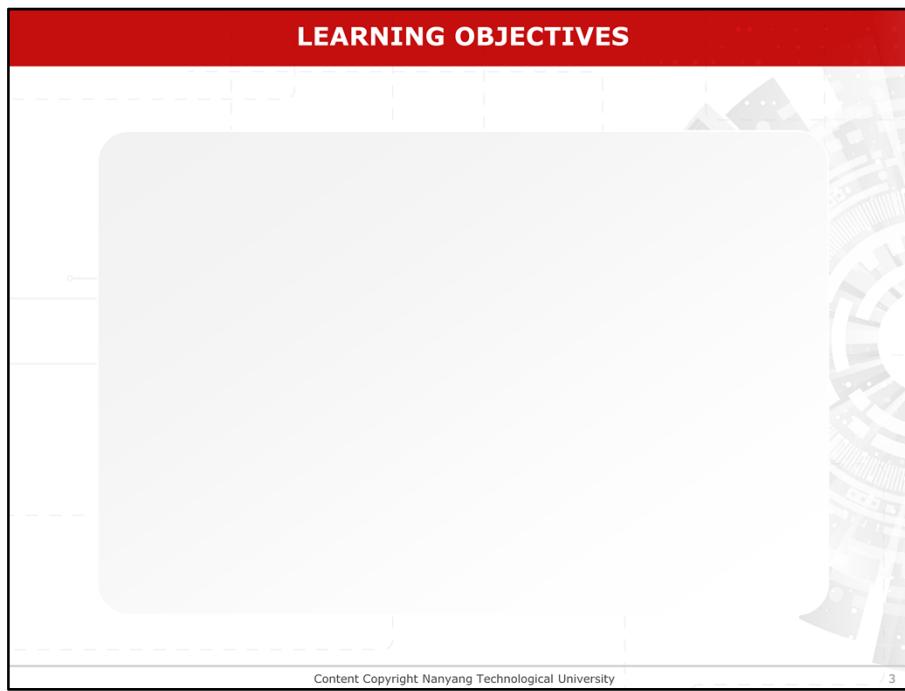
- Why do we Learn C Programming Language?
- Development of a C Program
- Data Type, Constants, Variables and Operators
- Simple Input and Output

Content Copyright Nanyang Technological University

2

Basic C Programming

This video lesson focuses on simple input and output operations



[Pause 1 sec]

LEARNING OBJECTIVES

After this lesson, you should be able to:

Content Copyright Nanyang Technological University 4

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

LEARNING OBJECTIVES

After this lesson, you should be able to:

- Explain input and output functions

Content Copyright Nanyang Technological University 5

Explain input and output functions.

[Pause 1 sec]

LEARNING OBJECTIVES

After this lesson, you should be able to:

- Explain input and output functions
- Explain the application of printf() with examples

Content Copyright Nanyang Technological University 6

Explain the application of printf() with examples.

[Pause 1 sec]

LEARNING OBJECTIVES

After this lesson, you should be able to:

- Explain input and output functions
- Explain the application of printf() with examples
- Explain the application of scanf() with examples

Content Copyright Nanyang Technological University 7

Explain the application of scanf() with examples.

[Pause 1 sec]

LEARNING OBJECTIVES

After this lesson, you should be able to:

- Explain input and output functions
- Explain the application of printf() with examples
- Explain the application of scanf() with examples
- Identify common errors in writing C programs

Content Copyright Nanyang Technological University

Identify common errors in writing C programs.

[Pause 1 sec]

SIMPLE INPUT/OUTPUT

- Most programs need to communicate with their environment.

Content Copyright Nanyang Technological University 9

Simple Input/Output

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

SIMPLE INPUT/OUTPUT

- Most programs need to communicate with their environment.
- Input/output (I/O) is the way a program communicates with the user.

Content Copyright Nanyang Technological University 10

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 I/O functions in the standard I/O libraries.

Content Copyright Nanyang Technological University 11

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.

Content Copyright Nanyang Technological University 12

Simple Input/Output

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

SIMPLE INPUT/OUTPUT

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

- `printf()`: Formatted output function
- `scanf()`: Formatted input function

Content Copyright Nanyang Technological University 13

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

SIMPLE INPUT/OUTPUT

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

- `printf()`: Formatted output function
- `scanf()`: Formatted input function
- `putchar()`: Character output function
- `getchar()`: Character input function

Content Copyright Nanyang Technological University 14

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

SIMPLE INPUT/OUTPUT

The following two Input/output functions are most frequently used:

- **printf()**: Formatted output function
- **scanf()**: Formatted input function

The diagram shows a flow of data from a 'Keyboard' to a 'Program' and then to a 'Display'. A green arrow labeled 'Data' points from the 'Keyboard' to the 'Program'. Above the arrow is the text 'scanf()'. Another green arrow labeled 'Data' points from the 'Program' to the 'Display'. Above this arrow is the text 'printf()'. The 'Program' is represented by a yellow box labeled 'Program'.

Content Copyright Nanyang Technological University 15

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,

SIMPLE INPUT/OUTPUT

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

Content Copyright Nanyang Technological University 16

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

SIMPLE INPUT/OUTPUT

- A function is a piece of code to perform a specific task.
- A library contains a group of functions, usually for related tasks.

Content Copyright Nanyang Technological University 17

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

SIMPLE INPUT/OUTPUT

- A function is a piece of code to perform a specific task.
- A library contains a group of functions, usually for related tasks.
- The I/O functions are in the C library `<stdio>`, to use the I/O functions, we need to include the header file:

`#include <stdio.h>`

as the **preprocessor instruction** in a program.

Content Copyright Nanyang Technological University 18

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.

Content Copyright Nanyang Technological University 19

The printf() Function

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

THE PRINTF() FUNCTION

- The **printf()** function is the most commonly used C library function.
- It allows us to control the format of the output.

Content Copyright Nanyang Technological University 20

The printf() Function

It allows us to control the format of the output.

THE **PRINTF()** FUNCTION

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

Content Copyright Nanyang Technological University

21

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

The printf() statement has the form:

```
printf(control-string, argument-list);
```

Content Copyright Nanyang Technological University 22

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

THE PRINTF() FUNCTION

The printf() statement has the form:

```
printf(control-string, argument-list);
```

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

Content Copyright Nanyang Technological University 23

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

THE PRINTF() FUNCTION

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;
}
```

Content Copyright Nanyang Technological University 24

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

THE PRINTF() FUNCTION

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

Content Copyright Nanyang Technological University 25

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() FUNCTION

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

Content Copyright Nanyang Technological University 26

The printf statement has the form as shown here.

THE printf() FUNCTION

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;
}
```

Content Copyright Nanyang Technological University 27

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

Content Copyright Nanyang Technological University 28

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**.

CONTROL-STRING CONVERSION SPECIFICATION

A **conversion specification** is of the form:

% [flag][minimumFieldWidth][.precision]conversionSpecifier

Content Copyright Nanyang Technological University

29

Conversion Specification

The format of a conversion specification is shown here.

CONTROL-STRING CONVERSION SPECIFICATION

A **conversion specification** is of the form:

% [flag][minimumFieldWidth][.precision]conversionSpecifier

- **%** and **conversionSpecifier** are compulsory. The others are optional.

Content Copyright Nanyang Technological University 30

Conversion Specification

The format of a conversion specification is shown here.

CONTROL-STRING CONVERSION SPECIFICATION

A **conversion specification** is of the form:

% [flag][minimumFieldWidth][.precision]conversionSpecifier

- **%** and **conversionSpecifier** are compulsory. The others are optional.

The **conversion Specifier** specifies how the data is to be converted into displayable form.

Content Copyright Nanyang Technological University 31

Conversion Specification

The format of a conversion specification is shown here.

CONTROL-STRING CONVERSION SPECIFICATION

A **conversion specification** is of the form:

% [flag][minimumFieldWidth][.precision]conversionSpecifier

- **%** and **conversionSpecifier** are compulsory. The others are optional.

It consists of a conversion specifier that indicates the type of required conversion to be performed.

Content Copyright Nanyang Technological University 32

Conversion Specification

The format of a conversion specification is shown here.

CONTROL-STRING CONVERSION SPECIFICATION

Note:

- We will mainly focus on using the compulsory options **%** and **conversionSpecifier**.

Content Copyright Nanyang Technological University 33

Conversion Specification

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

CONTROL-STRING CONVERSION SPECIFICATION

Note:

- We will mainly focus on using the compulsory options **%** and **conversionSpecifier**.
- Please refer to your textbook for the other options such as **flag**, **minimumFieldWidth** and **precision**.

Content Copyright Nanyang Technological University

34

Conversion Specification

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

The diagram shows a slide titled "CONTROL-STRING CONVERSION SPECIFICATION". The slide content is as follows:

`% [flag][minimumFieldWidth].[precision]conversionSpecifier`

The **flag** field 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.

Content Copyright Nanyang Technological University 35

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 diagram shows a slide titled "CONTROL-STRING CONVERSION SPECIFICATION". The slide contains a code snippet: "% [flag][minimumFieldWidth].[precision]conversionSpecifier". The word "minimumFieldWidth" is highlighted with a red border. Below the code, a callout box contains the following text: "The **minimumFieldWidth** field gives the minimum field width to be used during printing. For example, "%10d" prints an integer with field width of 10." The slide has a footer with the text "Content Copyright Nanyang Technological University" and the number "36".

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

CONTROL-STRING CONVERSION SPECIFICATION

% [flag][minimumFieldWidth][.precision]conversionSpecifier

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

For example, "%**8.4f**" prints a floating point number in a field of 8 characters wide with 4 digits after the decimal point.

Content Copyright Nanyang Technological University 37

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

PRINTF() – CONVERSION SPECIFIER

Some common types of *Conversion Specifiers*:

d	signed decimal conversion of int
o	unsigned octal conversion of unsigned
x,X	unsigned hexadecimal conversion of unsigned
c	single character conversion
f	signed decimal floating point conversion
s	string conversion

Content Copyright Nanyang Technological University

38

Conversion Specifier

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

PRINTF() – CONVERSION SPECIFIER

Other *Conversion Specifiers*:

e, E, g, G	Floating point number in decimal or scientific format
%	% symbol
p	A pointer

Content Copyright Nanyang Technological University

39

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;
}
```

Content Copyright Nanyang Technological University 40

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

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
int num = 10

Content Copyright Nanyang Technological University 41

The first printf() statement prints an integer.

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

```
float i = 10.300000
double j = 100.345600
```

Content Copyright Nanyang Technological University 42

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

Content Copyright Nanyang Technological University 43

The fourth **printf()** statement print the numbers with precision 2, i.e. only two 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

Content Copyright Nanyang Technological University 44

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.

SIMPLE INPUT: SCNF()

- A scanf() statement has the form:
 - **scanf (control-string, argument-list);**
- **Control-string** is a string constant containing conversion specifications.

Content Copyright Nanyang Technological University 45

The scanf() Function

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

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

SIMPLE INPUT: SCNF()

- The **argument-list** contains a list of items.

Content Copyright Nanyang Technological University 46

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

SIMPLE INPUT: SCNF()

- The **argument-list** contains a list of items.
 - The **items** in `scanf()` may be any **variable** matching the type given by the conversion specification.

Content Copyright Nanyang Technological University

47

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

SIMPLE INPUT: SCNF()

- The **argument-list** contains a list of items.
 - The **items** in `scanf()` may be any **variable** matching the type given by the conversion specification.
 - It cannot be a constant. It cannot be an expression like `n1 + n2`.

Content Copyright Nanyang Technological University

48

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

SIMPLE INPUT: SCNF()

- The **argument-list** contains a list of items.
 - The **items** in `scanf()` may be any **variable** matching the type given by the conversion specification.
 - It cannot be a constant. It cannot be an expression like `n1 + n2`.
 - The **variable name** has to be preceded by an **&**. 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.

Content Copyright Nanyang Technological University

49

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.

SIMPLE INPUT: SCNF()

- The **argument-list** contains a list of items.
 - The **items** in `scanf()` may be any **variable** matching the type given by the conversion specification.
 - It cannot be a constant. It cannot be an expression like `n1 + n2`.
 - The **variable name** has to be preceded by an **&**. 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.
- `scanf()` **stops reading** when it has read **all** the items as indicated by the control string or the **EOF** (end of file) is encountered.

Content Copyright Nanyang Technological University 50

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.

SIMPLE INPUT: SCNF()

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

Content Copyright Nanyang Technological University

51

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

SIMPLE INPUT: SCNF()

- `scanf()` uses whitespace characters (tabs, spaces and newlines) to determine how to separate the input into different fields to be stored.
- It ignores consecutive whitespace characters in between when matching up consecutive conversion specifications to different consecutive fields.

Content Copyright Nanyang Technological University

52

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

SIMPLE INPUT: SCNF()

- `scanf()` uses whitespace characters (tabs, spaces and newlines) to determine how to separate the input into different fields to be stored.
- It ignores consecutive whitespace characters in between when matching up consecutive conversion specifications to different consecutive fields.
- Conversion specification that begins with a percent sign (%) reads characters from input, converts the characters into values of the specified format, and stores the values to the address memory locations of the specified variables.

Content Copyright Nanyang Technological University

53

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

The diagram shows a slide titled "CONTROL-STRING CONVERSION SPECIFICATION". The slide contains a code snippet: "% [flag][minimumFieldWidth][.precision]conversionSpecifier". The word "conversionSpecifier" is highlighted with a red border. Below the code, a note states: "Where % and conversionSpecifier are compulsory. The others are optional." The slide has a red header bar and a white background with a faint watermark of a gear and text.

Content Copyright Nanyang Technological University

54

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.

SCANF() – CONVERSION SPECIFIER

Some common types of *Conversion Specifiers*:

d	signed decimal conversion of int
o	unsigned octal conversion of unsigned
x,X	unsigned hexadecimal conversion of unsigned
c	single character conversion
f	signed decimal floating point conversion
s	string conversion

Content Copyright Nanyang Technological University 55

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.

PRINTF() – CONVERSION SPECIFIER

Other common *Conversion Specifiers*:

e, E, g, G	Floating point number in decimal or scientific format
i	Integer number in decimal, octal or hexadecimal integer format
p	A pointer

Content Copyright Nanyang Technological University

56

Other conversion specifiers include:

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

PRINTF() – CONVERSION SPECIFIER

Other common *Conversion Specifiers*:

e, E, g, G	Floating point number in decimal or scientific format
i	Integer number in decimal, octal or hexadecimal integer format
p	A pointer

Content Copyright Nanyang Technological University 57

Other conversion specifiers include:

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

PRINTF() – CONVERSION SPECIFIER

Other common *Conversion Specifiers*:

e, E, g, G	Floating point number in decimal or scientific format
i	integer number in decimal, octal or hexadecimal integer format
p	A pointer

Content Copyright Nanyang Technological University

58

Other conversion specifiers include:

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

SIMPLE INPUT: SCNF()

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

Content Copyright Nanyang Technological University 59

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

SIMPLE INPUT: SCNF()

- 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.

Content Copyright Nanyang Technological University 60

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

SIMPLE INPUT: SCNF()

- 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 then starts reading the characters until it encounters a whitespace character.

Content Copyright Nanyang Technological University

61

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

SIMPLE INPUT: SCANF()

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

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

SIMPLE INPUT: SCANF()

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.

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

SIMPLE INPUT: SCNF()

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.

.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.

SIMPLE INPUT: SCNF()

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.

Content Copyright Nanyang Technological University

65

- 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;
}
```

Content Copyright Nanyang Technological University 66

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;
}
```

Content Copyright Nanyang Technological University

67



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

SCNF(): 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:

Content Copyright Nanyang Technological University

68

No VO

SCNF(): 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

Content Copyright Nanyang Technological University 69

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.

SCNF(): 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

Content Copyright Nanyang Technological University 70

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:



Content Copyright Nanyang Technological University

71

NO VO

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



Content Copyright Nanyang Technological University

72

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.

SCNF(): 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

Content Copyright Nanyang Technological University 73

THE SUM IS PRINTED ON THE SCREEN

SCNF(): 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;
}
```

Content Copyright Nanyang Technological University 74

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

SCNF(): EXAMPLE

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

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

    printf("Please enter 2 integers:\n");
    count = 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;
}
```

Content Copyright Nanyang Technological University 75

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`.

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 %f", 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?



Content Copyright Nanyang Technological University 76

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 `scanf()` 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 %f", &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?

Content Copyright Nanyang Technological University 77

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

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;
}
```

Content Copyright Nanyang Technological University

78

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:

Content Copyright Nanyang Technological University 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>

Content Copyright Nanyang Technological University 80

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



Content Copyright Nanyang Technological University

81

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**



Content Copyright Nanyang Technological University 82

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**



Content Copyright Nanyang Technological University 83

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

Content Copyright Nanyang Technological University 84

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?

Content Copyright Nanyang Technological University

85

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

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;
}
```

When the program runs:

Please enter a number:
1234<Enter>

The number read is 1234

Content Copyright Nanyang Technological University

86

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

COMMON ERROR 2: SCNF()

<pre>#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; }</pre>	<p>When the program runs:</p> <p>Please enter a number: 1234<Enter></p> <p>The number read is 1234 Correct (y/n)? Your reply:</p>
---	--

Content Copyright Nanyang Technological University

87

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

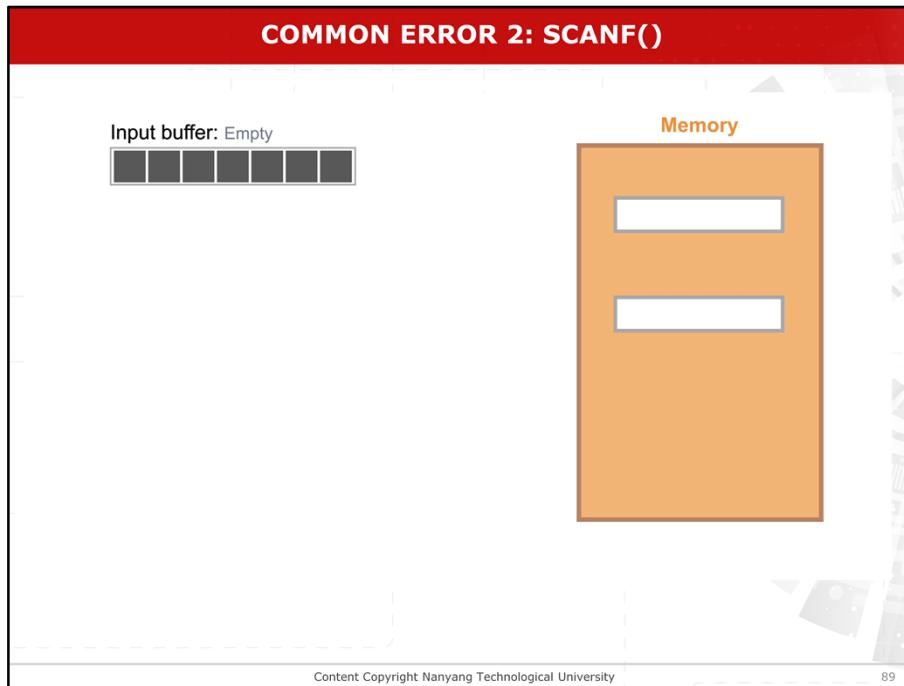
COMMON ERROR 2: SCNF()

<pre>#include <stdio.h> int main() { int number; char reply; printf("Please enter a number: "); scanf("%d", &number); // read in an <u>integer</u> printf("The number read is %d\n", number); printf("Correct (y/n)? "); scanf("%c", &reply); // read in a <u>char</u> printf("Your reply:%c\n", reply); // display the char -- an error return 0; }</pre>	When the program runs: <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">Please enter a number: 1234<Enter></div> <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc; border-radius: 5px; display: inline-block;">The number read is 1234 Correct (y/n)? Your reply:</div>
---	---

Content Copyright Nanyang Technological University

88

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.

COMMON ERROR 2: SUGGESTED SOLUTIONS

Solution 1: Read in '\n'

```
...
```

```
printf("Correct (y/n)?");
```

```
scanf("\n%c", &reply); // read the newline
```

```
printf("Your reply: %c\n", reply);
```

```
...
```

OR

```
char dummy;
```

```
...
```

```
scanf("%c",&dummy);
```

Content Copyright Nanyang Technological University

90

There are two ways to avoid this problem:

The first way is to modify the **scanf()** to read in a newline from the buffer.

COMMON ERROR 2: SUGGESTED SOLUTIONS**Solution 2: Using fflush()**

```
int number; char reply;  
printf("Enter a number: ");  
scanf("%d", &number); //read in an integer  
printf("The number read is %d\n", number);  
fflush(stdin); // flush the input buffer with newline  
printf("Correct (y/n)?");  
scanf("%c", &reply);  
printf("Your reply: %c\n", reply);
```

Content Copyright Nanyang Technological University

91

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.

COMMON ERROR 2: SUGGESTED SOLUTIONS**Solution 2: Using fflush()**

```
int number; char reply;  
  
printf("Enter a number: ");  
scanf("%d", &number); //read in an integer  
printf("The number read is %d\n", number);  
  
fflush(stdin); // flush the input buffer with newline  
printf("Correct (y/n)?");  
scanf("%c", &reply);  
printf("Your reply: %c\n", reply);
```

Content Copyright Nanyang Technological University

92

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.

COMMON ERROR 2: SUGGESTED SOLUTIONS**Solution 2: Using fflush()**

```
int number; char reply;  
  
printf("Enter a number: ");  
scanf("%d", &number); //read in an integer  
printf("The number read is %d\n", number);  
  
fflush(stdin); // flush the input buffer with newline  
printf("Correct (y/n)?");  
scanf("%c", &reply);  
printf("Your reply: %c\n", reply);
```

However, it is not suggested to use fflush() as it is not a standard C function. Some C compilers support fflush(), while others do not.

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.

CHARACTER INPUT/OUTPUT

There are two functions in the `<stdio>` library to manage single character input and output:

- `putchar()`
- `getchar()`

Content Copyright Nanyang Technological University

94

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

The slide has a red header bar with the text 'CHARACTER INPUT/OUTPUT'. The main content area contains the following text:
putchar()
• The syntax of calling putchar is:
putchar(characterConstantOrVariable);

Content Copyright Nanyang Technological University 95

putchar()

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

The slide has a red header bar with the title 'CHARACTER INPUT/OUTPUT'. The main content area contains the following text:

putchar()

- The syntax of calling putchar is:
`putchar(characterConstantOrVariable);`
- It is equivalent to:
`printf("%c", characterConstantOrVariable);`

Content Copyright Nanyang Technological University 96

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);`

CHARACTER INPUT/OUTPUT

putchar()

- The syntax of calling putchar is:
putchar(characterConstantOrVariable);
- It is equivalent to:
printf("%c", characterConstantOrVariable);
- The difference is that **putchar** is **faster** because printf needs to process the control string for formatting.

Content Copyright Nanyang Technological University 97

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

CHARACTER INPUT/OUTPUT

putchar()

- The syntax of calling putchar is:
`putchar(characterConstantOrVariable);`
- It 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.

Content Copyright Nanyang Technological University 98

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

The slide has a red header bar with the text 'CHARACTER INPUT/OUTPUT'. The main content area contains the following text:

getchar()

- The syntax of calling getchar is:

```
ch = getchar(); // ch is a character variable.
```

Content Copyright Nanyang Technological University 99

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()**.

The slide has a red header bar with the title "CHARACTER INPUT/OUTPUT". The main content area contains the following text:

getchar()

- The syntax of calling getchar is:
`ch = getchar(); // ch is a character variable.`
- It is equivalent to
`scanf("%c", &ch);`

At the bottom of the slide, there is a footer with the text "Content Copyright Nanyang Technological University" and the number "100".

It is equivalent to the statement shown here.

CHARACTER INPUT/OUTPUT

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

Content Copyright Nanyang Technological University 101

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

CHARACTER INPUT/OUTPUT

- 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.

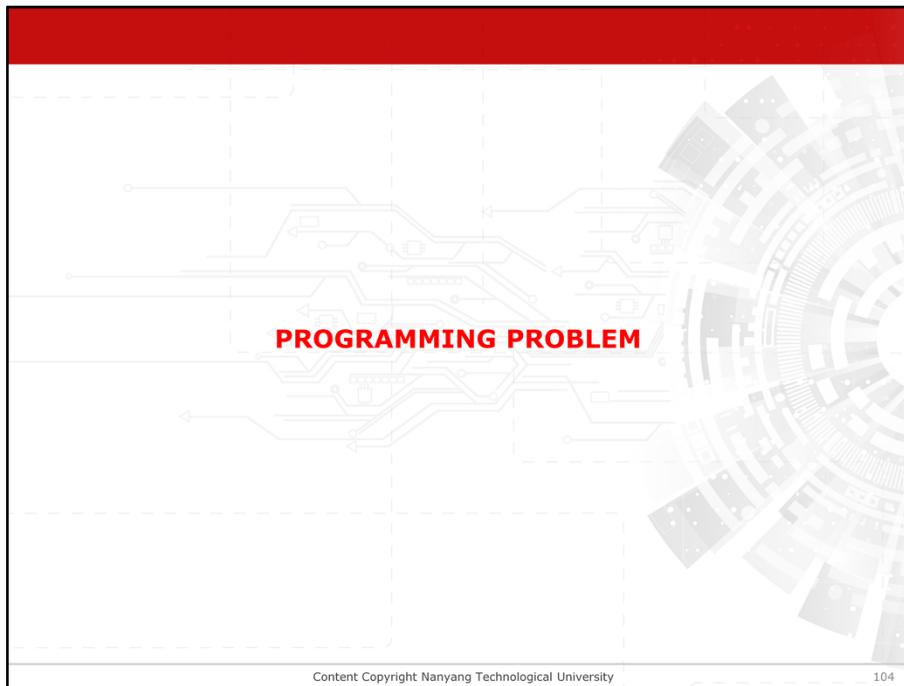
Content Copyright Nanyang Technological University 102

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

CHARACTER INPUT/OUTPUT

- 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.

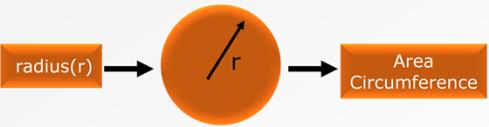
A *buffer position indicator* is used to keep track of the position where the data is read from the buffer.



**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
    → /* Write your code here */
    // Calculate the area
    → /* Write your code here */
    // Calculate the circumference
    → /* Write your code here */
    // Print the area and circumference of the circle
    → /* Write your code here */
    return 0;
}
```



Area = πr^2
Circumference = $2\pi r$

Content Copyright Nanyang Technological University

105

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("%f", &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;
}

```

The diagram illustrates the sequential structure of the program. It starts with a box labeled "radius(r)" with an arrow pointing to a circle containing "r". Another arrow points from the circle to a box labeled "Area Circumference".

Area = πr^2
Circumference = $2\pi r$

Output
Enter the radius: 5.0

Content Copyright Nanyang Technological University 106

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("%f", &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;
}
```

The diagram illustrates the sequential structure of the program. It starts with a box labeled "radius(r)" with an arrow pointing to a circle containing "r". Another arrow points from the circle to a box labeled "Area Circumference". To the right of the circle, the formulas "Area = π*r*r" and "Circumference= 2*π*r" are shown. Below the circle is a box labeled "Output" containing the text "Enter the radius: 5.0".

Content Copyright Nanyang Technological University 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("%f", &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 = πr^2
Circumference = $2\pi r$

Output

Enter the radius: 5.0
The area is 78.500000

Content Copyright Nanyang Technological University

108

and displays the area and circumference on the screen.

**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("%f", &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;
}
```

radius(r) → → Area Circumference

Area = $\pi \cdot r \cdot r$
Circumference = $2 \cdot \pi \cdot r$

Output

```
Enter the radius: 5.0
The area is 78.500000
The circumference is 31.400000
```

Content Copyright Nanyang Technological University

109

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.

SUMMARY

After this lesson, you should be able to:

- Explain input and output functions
- Explain the application of printf() with examples
- Explain the application of scanf() with examples
- Identify common errors in writing C programs

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