

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**

There are 4 main sections to cover for Basic C Programming as shown.

This video lesson focuses on the third part: Data types, constants, variables and operations.

**LEARNING OBJECTIVES**

After this lesson, you should be able to:

Content Copyright Nanyang Technological University 3

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

**LEARNING OBJECTIVES**

After this lesson, you should be able to:

- Differentiate computer memory and variables

Content Copyright Nanyang Technological University 4

Differentiate computer memory and variables

## LEARNING OBJECTIVES

After this lesson, you should be able to:

- Differentiate computer memory and variables
- Explain data and its types

Content Copyright Nanyang Technological University 5

Explain Data and its types.  
[Pause 1 sec]

## LEARNING OBJECTIVES

After this lesson, you should be able to:

- Differentiate computer memory and variables
- Explain data and its types
- Define constants

Content Copyright Nanyang Technological University 6

Define constants.

[Pause 1 sec]

## LEARNING OBJECTIVES

After this lesson, you should be able to:

- Differentiate computer memory and variables
- Explain data and its types
- Define constants
- Define variables

Content Copyright Nanyang Technological University 7

Define variables  
[Pause 1 sec]

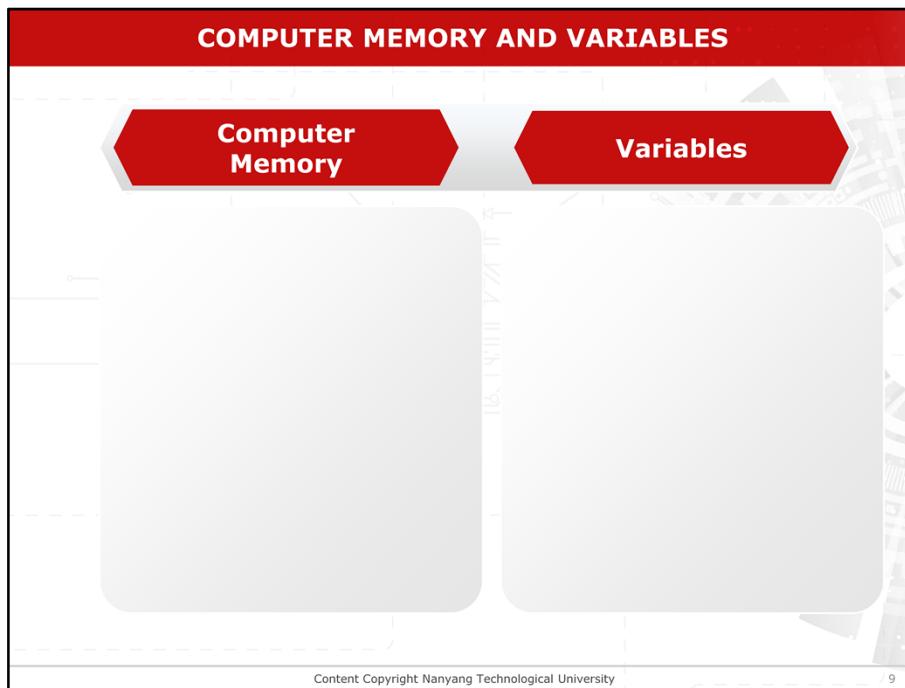
## LEARNING OBJECTIVES

After this lesson, you should be able to:

- Differentiate computer memory and variables
- Explain data and its types
- Define constants
- Define variables
- Explain the use of operators

Content Copyright Nanyang Technological University 8

explain the use of operators.  
[Pause 1 sec]



## Computer Memory and Variables

**COMPUTER MEMORY AND VARIABLES**

**Computer Memory**

- Used for **storing** data objects.

**Variables**

Content Copyright Nanyang Technological University 10

### **Computer Memory and Variables**

A program needs to work with *data* which are stored in the main memory.

**COMPUTER MEMORY AND VARIABLES**

**Computer Memory**

- Used for **storing** data objects.
- Data object can be a variable or constant.

**Variables**

Content Copyright Nanyang Technological University 11

A data object in a C program can be a variable or a constant.

The diagram is titled "COMPUTER MEMORY AND VARIABLES" in a red header bar. It features two main sections: "Computer Memory" on the left and "Variables" on the right, each with a list of bullet points. The "Computer Memory" section lists: "Used for **storing** data objects." and "Data object can be a variable or constant." The "Variables" section lists: "Used for **storing** data whose values change during the execution of a program." At the bottom left is the copyright notice "Content Copyright Nanyang Technological University" and at the bottom right is the number "12".

**COMPUTER MEMORY AND VARIABLES**

**Computer Memory**

- Used for **storing** data objects.
- Data object can be a variable or constant.

**Variables**

- Used for **storing** data whose values change during the execution of a program.

Content Copyright Nanyang Technological University 12

Variables are used to store data whose values change during the execution of a program, while constants can be used to store data whose values remain fixed during the execution of a program.

**COMPUTER MEMORY AND VARIABLES**

- Computer memory is used for storing data objects (variables or constants).

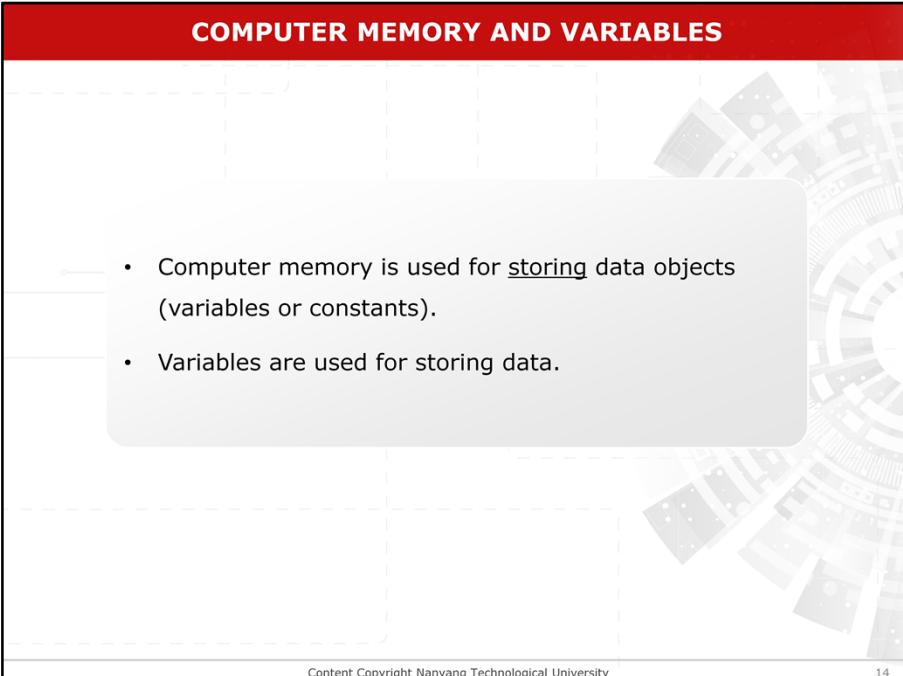
Content Copyright Nanyang Technological University

13

### **Computer Memory and Variables**

Computer memory is used for storing data objects (variables or constants).

**COMPUTER MEMORY AND VARIABLES**



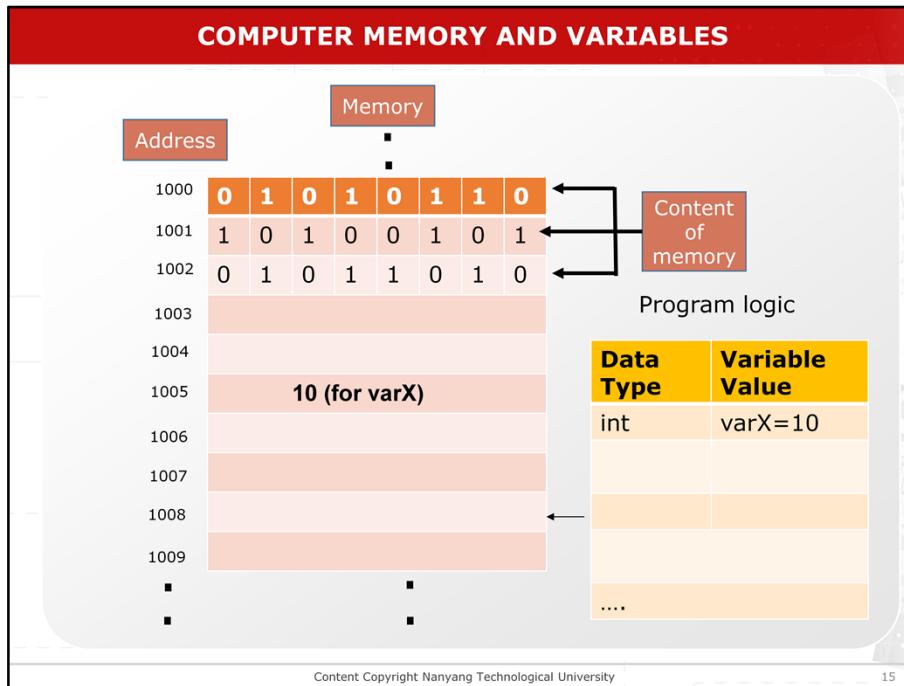
Content Copyright Nanyang Technological University

14

- Computer memory is used for storing data objects (variables or constants).
- Variables are used for storing data.

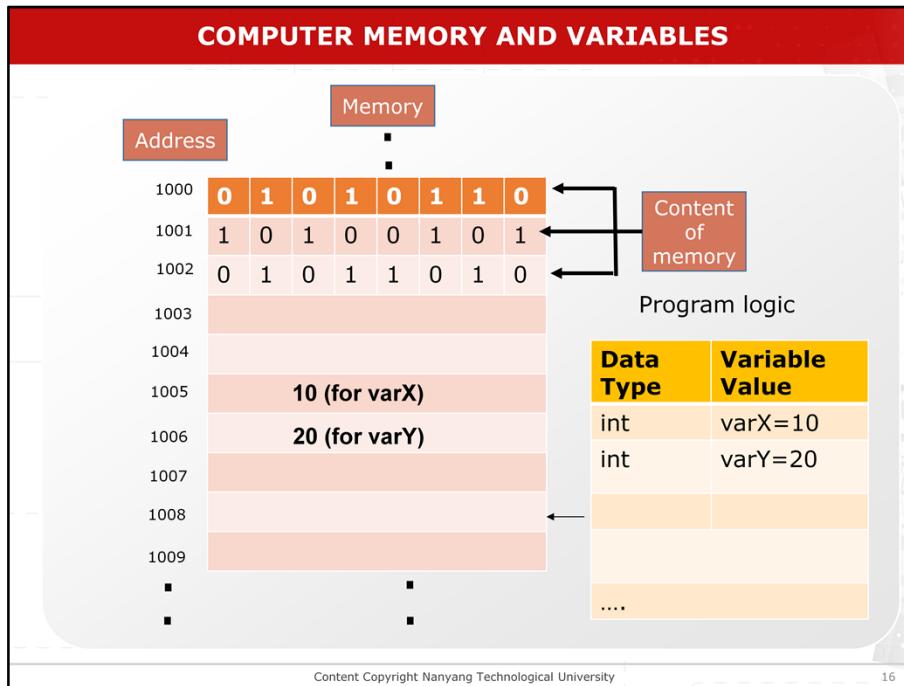
### **Computer Memory and Variables**

Variables are used for storing data.

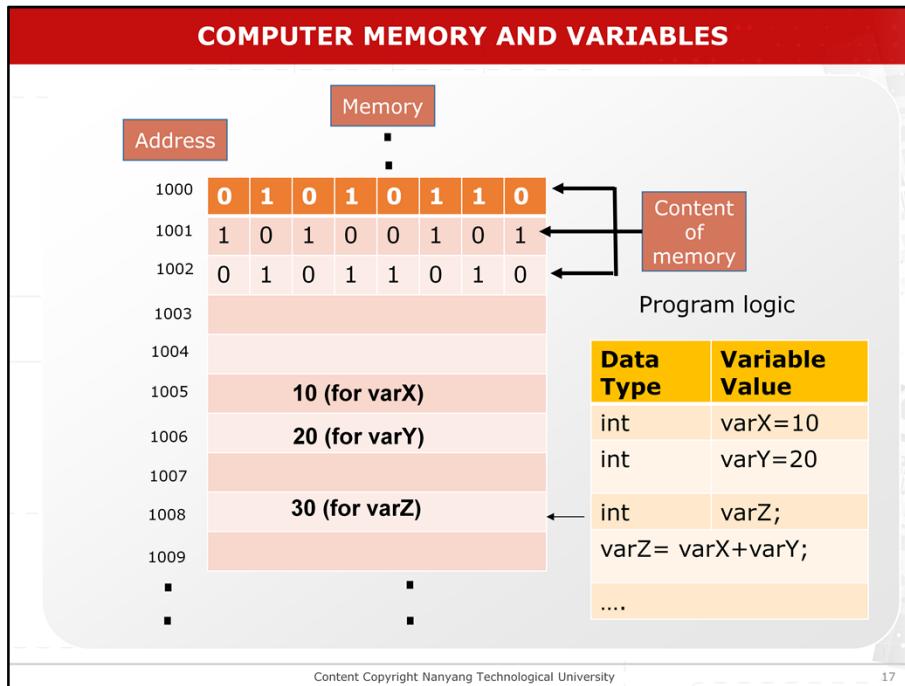


### Computer Memory and Variables

A program needs to work with *data* which are stored in the main memory



Program logic is shown here.



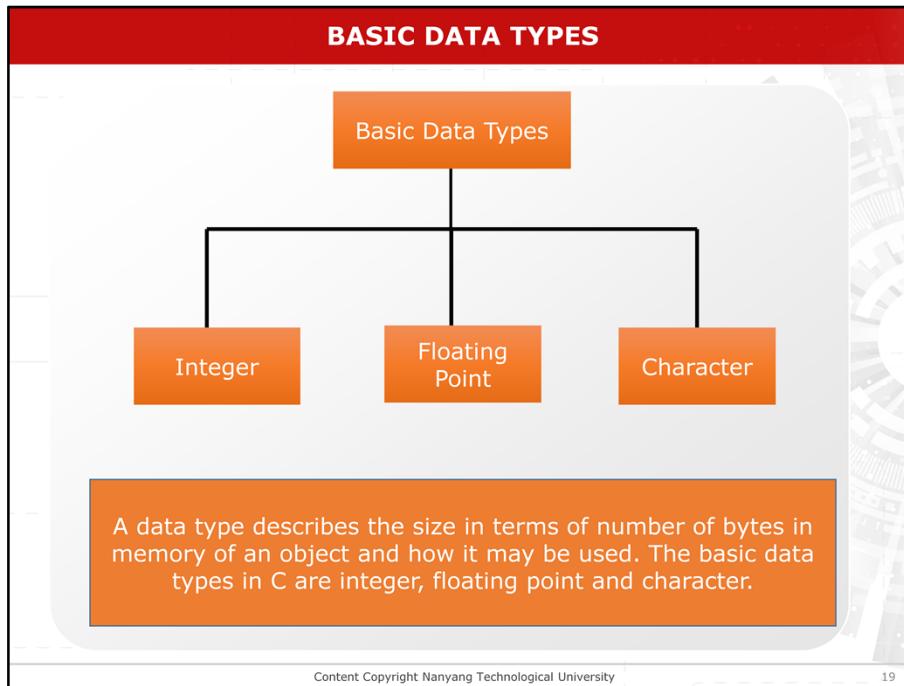
Both constants and variables must be of a certain type.

**BASIC DATA TYPES**

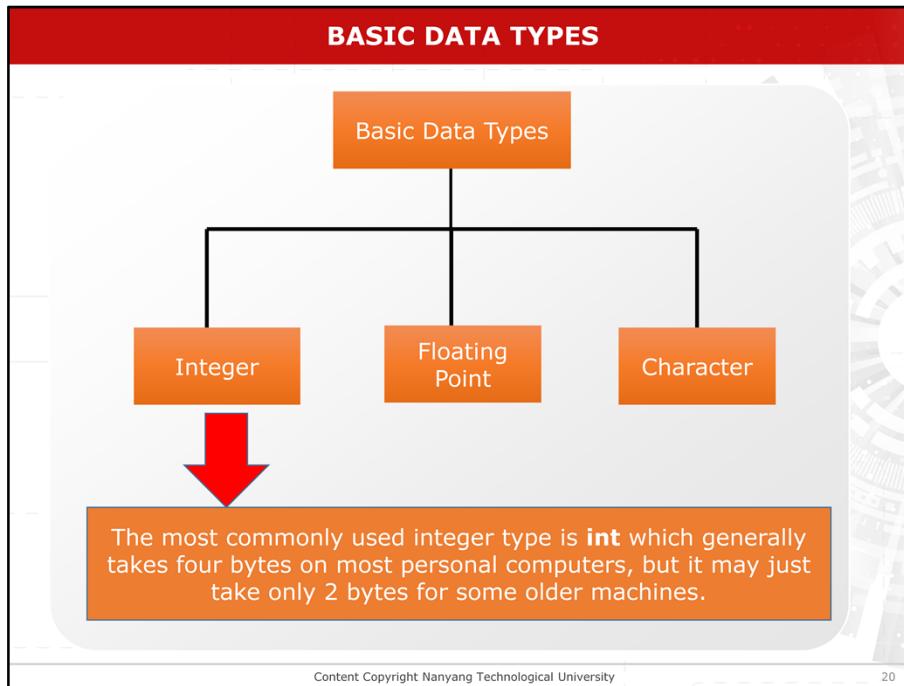
Content Copyright Nanyang Technological University

18

The C language provides a variety of data types for storing data for the needs of different programs.



A data type describes the size in terms of number of bytes in memory of an object and how it may be used. The basic data types in C are integer, floating point and character. Each type has its own computational properties and memory requirements. Moreover, the same data type may take different sizes on different types of machines.



In C, there are six ways to store an integer object in a computer for C: short, int, long, unsigned, unsigned short, unsigned long. The most commonly used integer data type is **int** which generally takes four bytes on most personal computers, but it may just take only 2 bytes for some older machines.

## FLOATING POINTS

- Any data object, which is a real number, is in this category.

Content Copyright Nanyang Technological University

21

Any data object, which is a real number, is in this category.

## FLOATING POINTS

- Any data object, which is a real number, is in this category.
- Floating point number representation is similar to scientific notation that is especially useful in expressing very small or very large numbers.

Content Copyright Nanyang Technological University

22

Floating point number representation is similar to scientific notation that is especially useful in expressing very small or very large numbers.

## FLOATING POINTS

- Any data object, which is a real number, is in this category.
- Floating point number representation is similar to scientific notation that is especially useful in expressing very small or very large numbers.
- Exponential notation is used by most computers to represent floating point numbers.

Content Copyright Nanyang Technological University

23

Exponential notation is used by most computers to represent floating point numbers.

## FLOATING POINTS

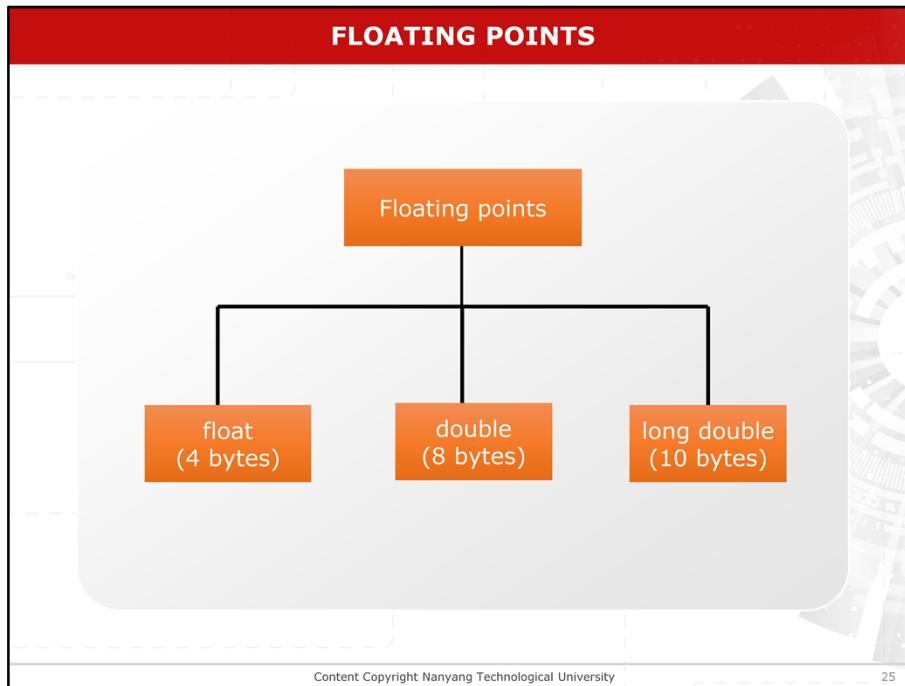
- Any data object, which is a real number, is in this category.
- Floating point number representation is similar to scientific notation that is especially useful in expressing very small or very large numbers.
- Exponential notation is used by most computers to represent floating point numbers.
- There are three ways to store a floating point object for C in the computer.

Content Copyright Nanyang Technological University

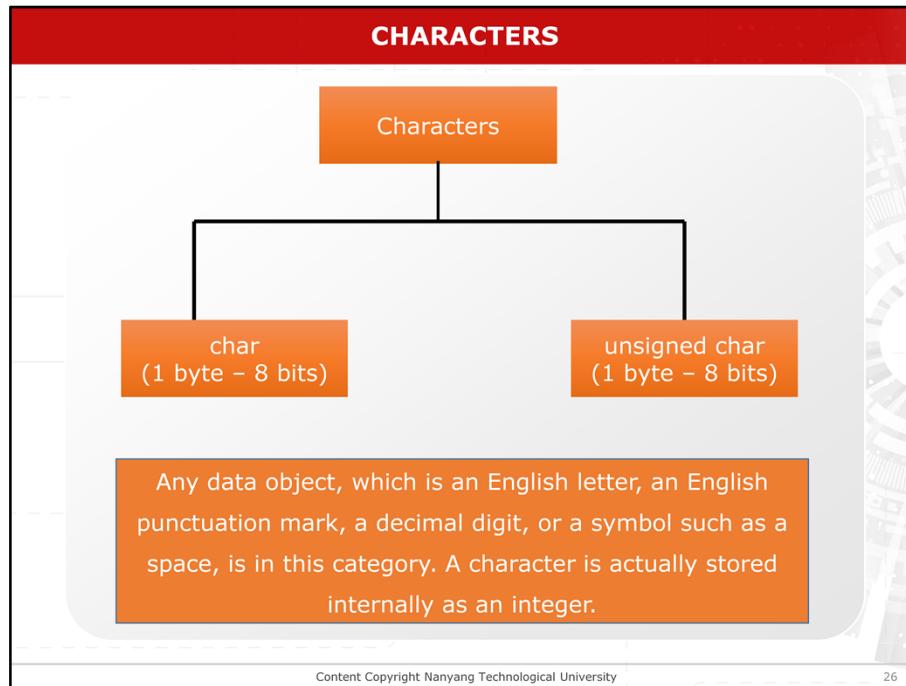
24

### Floating Point Type

There are three ways to store a floating point object for C in the computer.

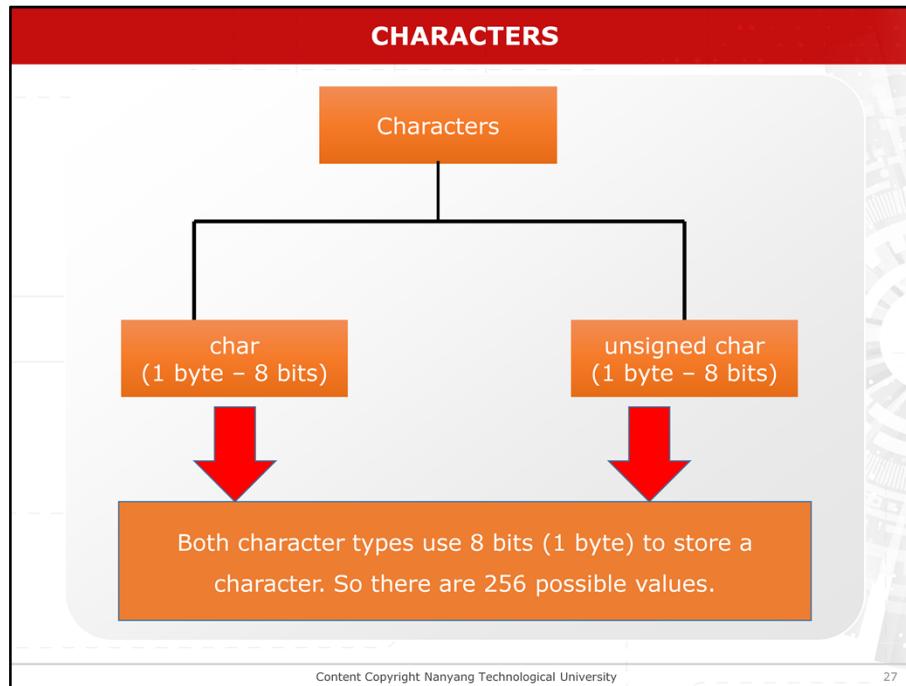


Therefore, three C floating point types are available: float, double and long double.



### Character Type

Any data object, which is an English letter, an English punctuation mark, a decimal digit, or a symbol such as a space, is in this category. A character is actually stored internally as an integer.



### Character Type

There are two C character types: **char** and **unsigned char**. Both character types use 8 bits (1 byte) to store a character. So there are 256 possible values.

CHARACTER - ASCII SET (1 BYTE)											
	0	1	2	3	4	5	6	7	8	9	
0	NUL							BEL	BS	TAB	
1	LF		FF	CR							
2								ESC			
3			SP	!	"	#	\$	%	&	'	
4	(	)	*	+	,	-	.	/	0	1	
5	2	3	4	5	6	7	8	9	:	;	
6	<	=	>	?	@	A	B	C	D	E	
7	F	G	H	I	J	K	L	M	N	O	
8	P	Q	R	S	T	U	V	W	X	Y	
9	Z	[	\	]	^	_	'	a	b	c	
10	d	e	f	g	h	i	j	k	l	m	
11	n	o	p	q	r	s	t	u	v	w	
12	x	y	z	{		}	~	DEL			

Content Copyright Nanyang Technological University

28

A character is actually stored internally as an integer. The conversion from characters to integer numbers is done commonly using the ASCII code. The ASCII (American Standard Code Information Interchange) code is the most commonly used character set. There are 128 distinct characters in the ASCII character set.

CHARACTER - ASCII SET (1 BYTE)										
	0	1	2	3	4	5	6	7	8	9
0	NUL						BEL	BS	TAB	
1	LF		FF	CR						
2							ESC			
3			SP	!	"	#	\$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	DEL		

Content Copyright Nanyang Technological University

29

## Characters

Character constants can be given by quoting the numerical value of a character, e.g. 65 for the character A in the ASCII character set, or by enclosing it with quotes, e.g. 'A'.

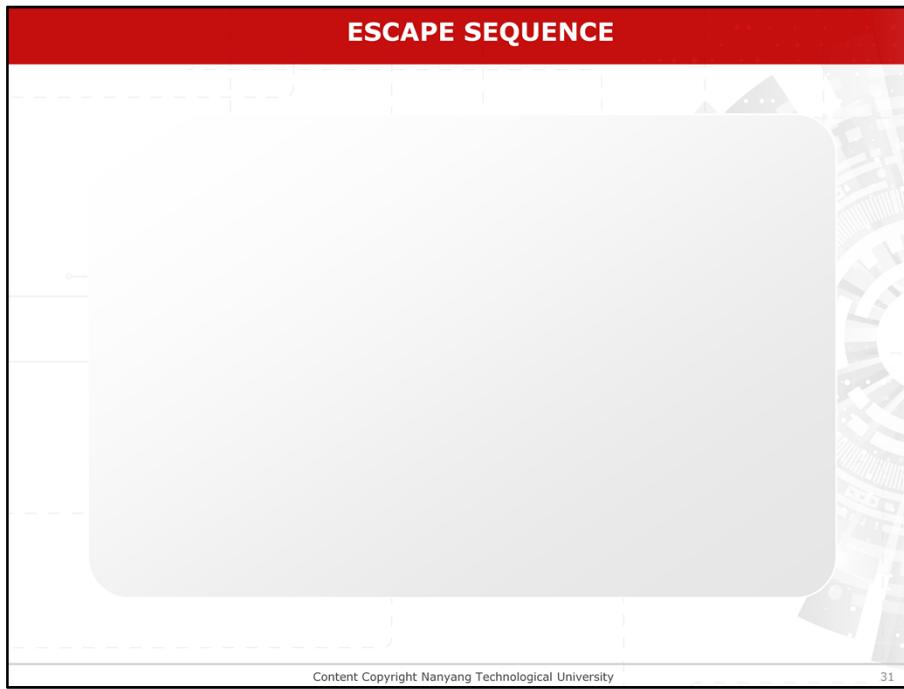
CHARACTER - ASCII SET (1 BYTE)											
	0	1	2	3	4	5	6	7	8	9	
0	NUL							BEL	BS	TAB	
1	LF		FF	CR							
2								ESC			
3			SP	!	"	#	\$	%	&	'	
4	(	)	*	+	,	-	.	/	0	1	
5	2	3	4	5	6	7	8	9	:	;	
6	<	=	>	?	@	A	B	C	D	E	
7	F	G	H	I	J	K	L	M	N	O	
8	P	Q	R	S	T	U	V	W	X	Y	
9	Z	[	\	]	^	_	'	a	b	c	
10	d	e	f	g	h	i	j	k	l	m	
11	n	o	p	q	r	s	t	u	v	w	
12	x	y	z	{		}	~	DEL			

Content Copyright Nanyang Technological University

30

### Characters

A *string constant* is a sequence of characters enclosed in double quotation marks. It is different from a character constant.



### Escape Sequence

**ESCAPE SEQUENCE**

- Some useful **non-printable control characters** are referred to by the ***escape sequence***.

Content Copyright Nanyang Technological University 32

### Characters – Escape Sequence

Some useful non-printable control characters are referred to by the ***escape sequence***.

## ESCAPE SEQUENCE

- Some useful **non-printable control characters** are referred to by the ***escape sequence***.
- It consists of the backslash symbol (\) followed by a single character.

Content Copyright Nanyang Technological University

33

It consists of the backslash symbol followed by a single character. The two symbols together represent the single required character.

## ESCAPE SEQUENCE

- Some useful **non-printable control characters** are referred to by the ***escape sequence***.
- It consists of the blackslash symbol (\) followed by a single character.
- This is a better alternative, in terms of memorisation, than ASCII numbers.

Content Copyright Nanyang Technological University

34

This is a better alternative, in terms of memorization, than ASCII numbers.

## ESCAPE SEQUENCE

- Some useful **non-printable control characters** are referred to by the ***escape sequence***.
- It consists of the blackslash symbol (\) followed by a single character.
- This is a better alternative, in terms of memorisation, than ASCII numbers.
- Example: '\n' the newline (or linefeed) character instead of the number **10**.

Content Copyright Nanyang Technological University

35

For example, '\n' represents the newline character instead of using the ASCII number 10.

EXAMPLES OF ESCAPE SEQUENCE					
'\a'	alarm bell	'\f'	form feed	'\n'	newline
'\t'	horizontal tab	'\"'	double quote	'\v'	vertical tab
'\b'	back space	'\\'	backslash	'\r'	carriage return
'\''	single quote				

This table gives some examples of escape sequence. Notice that escape sequences must be enclosed in single quotes.

EXAMPLES OF ESCAPE SEQUENCE					
'\a'	alarm bell	'\f'	form feed	'\n'	newline
'\t'	horizontal tab	'\"'	double quote	'\v'	vertical tab
'\b'	back space	'\\'	backslash	'\r'	carriage return
'\''	single quote				

'\007' and '\x7' are octal and hexadecimal values for the alarm bell.

Content Copyright Nanyang Technological University

37

The formats for octal and hexadecimal values are '\0oo' and '\xhh'. For example, '\007' and '\x7' are octal and hexadecimal values for the alarm bell.

EXAMPLES OF ESCAPE SEQUENCE					
'\a'	alarm bell	'\f'	form feed	'\n'	newline
'\t'	horizontal tab	'\"'	double quote	'\v'	vertical tab
'\b'	back space	'\\'	backslash	'\r'	carriage return
'\''	single quote				

The character constant 'A' (may be represented as '\0101' or '\x41' for octal and hexadecimal values respectively.

Content Copyright Nanyang Technological University

38

The character constant 'A' (with ASCII encoding of decimal 65, octal 101 or hexadecimal 41) may be represented as '\0101' or '\x41' for octal and hexadecimal values respectively.

The slide has a red header bar with the word "CONSTANT" in white capital letters. Below the header is a large, semi-transparent grey callout box containing a single bullet point. The background of the slide features a faint watermark of a city skyline.

• A constant is an object whose value is unchanged (or cannot be changed) throughout the program execution.

Content Copyright Nanyang Technological University 39

### **Constants**

A constant is an object whose value is unchanged (or cannot be changed) throughout the program execution.

## CONSTANT

- A constant is an object whose value is unchanged (or cannot be changed) throughout the program execution.
- There are four types of constant values:

Content Copyright Nanyang Technological University 40

There are four types of constant values:

## CONSTANT

- A constant is an object whose value is unchanged (or cannot be changed) throughout the program execution.
- There are four types of constant values:
  - Integer: e.g. 100, -256

Content Copyright Nanyang Technological University

41

- Integer: e.g. 100, -256

## CONSTANT

- A constant is an object whose value is unchanged (or cannot be changed) throughout the program execution.
- There are four types of constant values:
  - Integer: e.g. 100, -256
  - Floating point: e.g. 2.4, -3.0

Content Copyright Nanyang Technological University 42

- floating point: e.g. 2.4, -3.0;

## CONSTANT

- A constant is an object whose value is unchanged (or cannot be changed) throughout the program execution.
- There are four types of constant values:
  - Integer: e.g. 100, -256
  - Floating point: e.g. 2.4, -3.0
  - Character: e.g. 'a', '+'

Content Copyright Nanyang Technological University 43

character: e.g. 'a', '+'

## CONSTANT

- A constant is an object whose value is unchanged (or cannot be changed) throughout the program execution.
- There are four types of constant values:
  - Integer: e.g. 100, -256
  - Floating point: e.g. 2.4, -3.0
  - Character: e.g. 'a', '+'
  - String constant: e.g. "have a good day"

Content Copyright Nanyang Technological University 44

string constants which are a string of characters: e.g. "have a good day".

## INTEGER CONSTANTS

Integer constants may be specified in decimal, octal or hexadecimal notation.

Integer constants may be specified in decimal, octal or hexadecimal notation.

**INTEGER CONSTANTS**

- Decimal integer constant (0-9), e.g. 1234

Content Copyright Nanyang Technological University

46

A decimal integer constant is written as a sequence of decimal digits (0-9), e.g. 1234.

## INTEGER CONSTANTS

- Decimal integer constant (0-9), e.g. 1234
- Octal integer constant (0-7) e.g. 077

Content Copyright Nanyang Technological University

47

An octal integer constant is written as a sequence of octal digits (0-7) starting with a zero (**0**), e.g. **077**.

## INTEGER CONSTANTS

- Decimal integer constant (0-9), e.g. 1234
- Octal integer constant (0-7) e.g. 077
- Hexadecimal integer constant (0-9) and (A to F) e.g. 0x or 0X or 0xFF

Content Copyright Nanyang Technological University 48

The hexadecimal digits comprise 0-9, and the letters A to F in uppercase or lowercase where the letters represent the values 10 to 15 respectively.

## INTEGER CONSTANTS

- Decimal integer constant (0-9), e.g. 1234
- Octal integer constant (0-7) e.g. 077
- Hexadecimal integer constant (0-9) and (A to F) e.g. 0x or 0X or 0xFF

A decimal, octal or hexadecimal integer constant can immediately be followed by the letter L or letter l to explicitly specify a long integer constant.  
e.g. 1234L and 0xFFL

Content Copyright Nanyang Technological University

49

A decimal, octal or hexadecimal integer constant can immediately be followed by the letter L uppercase or lowercase to explicitly specify a long integer constant, e.g. 1234L and 0xFFL

## FLOATING POINT CONSTANTS

- Floating point constants may be written in two forms.

Content Copyright Nanyang Technological University

50

Floating point constants may be written in two forms.

## FLOATING POINT CONSTANTS

- Floating point constants may be written in two forms.
- The first is in the form of a sequence of decimal digits including the decimal point, e.g. 3.1234.

Content Copyright Nanyang Technological University

51

The first is in the form of a sequence of decimal digits including the decimal point, e.g. 3.1234.

## FLOATING POINT CONSTANTS

- Floating point constants may be written in two forms.
- The first is in the form of a sequence of decimal digits including the decimal point, e.g. 3.1234.
- The second is to use the exponential notation, e.g.

**31234e3.**

Content Copyright Nanyang Technological University

52

The second is to use the exponential notation as discussed in the previous section, e.g. **31234e3.**

## FLOATING POINT CONSTANTS

- Floating point constants may be written in two forms.
- The first is in the form of a sequence of decimal digits including the decimal point, e.g. 3.1234.
- The second is to use the exponential notation, e.g.  
**31234e3.**
- All floating point constants are always stored as **double**. If the floating point numbers are qualified by the suffix **f** or **F**, e.g. **1.234F**, then the values are of type **float**.

Content Copyright Nanyang Technological University

53

All floating point constants are always stored as double. If the floating point numbers are qualified by the suffix f LOWERCASE OR UPPERCASE e.g. 1.234F, then the values are of type float

## FLOATING POINT CONSTANTS

- Floating point constants may be written in two forms.
- The first is in the form of a sequence of decimal digits including the decimal point, e.g. 3.1234.
- The second is to use the exponential notation, e.g.  
**31234e3.**
- All floating point constants are always stored as **double**. If the floating point numbers are qualified by the suffix **f** or **F**, e.g. **1.234F**, then the values are of type **float**.
- We may also qualify a floating point number with **I** or **L** to specify the value to be stored is of type **long double**.

Content Copyright Nanyang Technological University

54

We may also qualify a floating point number with **I** or **L** to specify the value to be stored is of type **long double**.

## CHARACTER CONSTANTS

Content Copyright Nanyang Technological University

55

Character constants

## CHARACTER CONSTANTS

- Character constants can be given by quoting the numerical value of a character, e.g. 65 for the character **A** in the ASCII character set, or by enclosing it with quotes, e.g. '**A**'.

- Character constants can be given by quoting the numerical value of a character, e.g. 65 for the character **eh** in the ASCII character set, or by enclosing it with quotes

## CHARACTER CONSTANTS

- Character constants can be given by quoting the numerical value of a character, e.g. 65 for the character **A** in the ASCII character set, or by enclosing it with quotes, e.g. 'A'.
- Some useful non-printable control characters are referred to by the *escape sequence* which consists of the backslash symbol (\) followed by a single character.

Content Copyright Nanyang Technological University

57

- Some useful non-printable control characters are referred to by the *escape sequence* which consists of the backslash symbol followed by a single character.

## CHARACTER CONSTANTS

- Character constants can be given by quoting the numerical value of a character, e.g. 65 for the character **A** in the ASCII character set, or by enclosing it with quotes, e.g. '**A**'.
- Some useful non-printable control characters are referred to by the *escape sequence* which consists of the backslash symbol (\) followed by a single character.
- The two symbols together represent the single required character.

Content Copyright Nanyang Technological University

58

- The two symbols together represent the single required character. This is a better alternative, in terms of memorization, than ASCII numbers.

## STRING CONSTANTS

- A *string* constant is a sequence of characters enclosed in double quotation marks. It is different from a character constant.

A *string* constant is a sequence of characters enclosed in double quotation marks. It is different from a character constant.

## STRING CONSTANTS

- A *string* constant is a sequence of characters enclosed in double quotation marks. It is different from a character constant.
- For example, 'a' and "a" are different as 'a' is a character while "a" is a string.

For example, 'eh' in single quotes and "eh" in double quotes are different as 'eh' in single quote is a character while "eh" in double quotes is a string. Strings will be discussed in details in the chapter on Character Strings



## Defining Constants

## DEFINING CONSTANTS

We can use the preprocessor directive for defining constants.

#define

### Defining Constants

We can use the preprocessor directive **#define as shown.**

## DEFINING CONSTANTS

The format of #define is

```
#define CONSTANT_NAME value
```

where **CONSTANT\_NAME** is the name of the constant.

Content Copyright Nanyang Technological University

63

### Defining Constants

The format of **#define** is shown here., where **CONSTANT\_NAME** is the name of the constant.

## DEFINING CONSTANTS

Examples:

```
#define YES 'Y'  
#define GREETINGS "How are you?"  
#define ALARM '\a'
```

Content Copyright Nanyang Technological University

64

Some examples are shown here.

## DEFINING CONSTANTS

Examples:

```
#define YES 'Y'  
#define GREETINGS "How are you?"  
#define ALARM '\a'
```

Content Copyright Nanyang Technological University

65

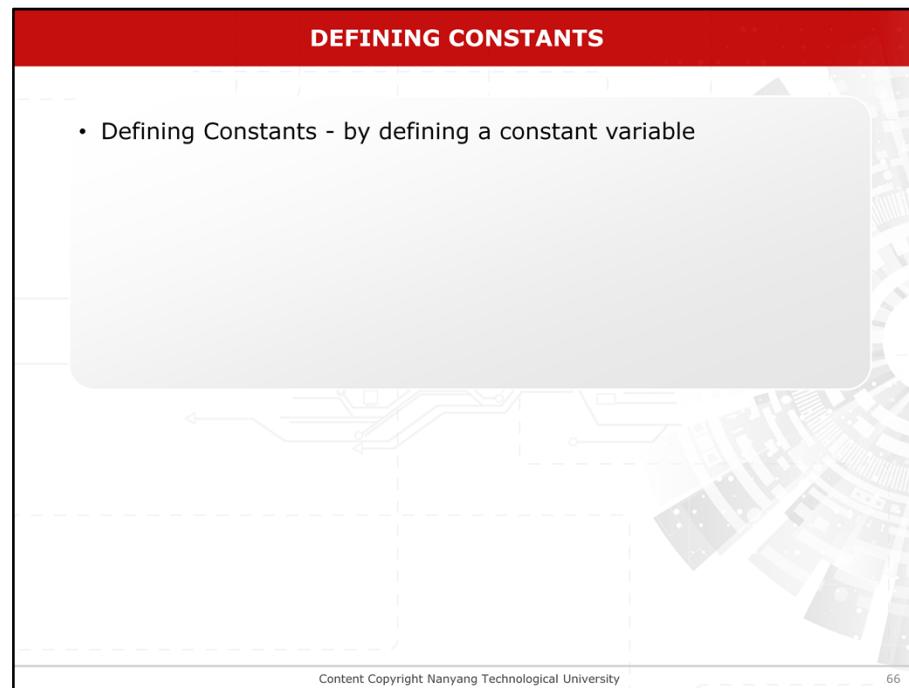
It is a C tradition that **CONSTANT\_NAME** is in upper case. It makes programs more readable. During compilation, the value of the constant will substitute the name of the constant wherever it appears in the program. By giving a symbolic name to a constant, it improves the readability of the program and makes the program easier to be modified.

**DEFINING CONSTANTS**

- Defining Constants - by defining a constant variable

Content Copyright Nanyang Technological University

66



Another way to define a constant is to use a constant variable.

## DEFINING CONSTANTS

- Defining Constants - by defining a constant variable

Format:

```
const type varName = value;
```

Content Copyright Nanyang Technological University 67

Format of defining constants is shown here.

The diagram shows a computer monitor with a white frame. On the screen, there is a red header bar with the text "DEFINING CONSTANTS" in white. Below the header, there is a list of bullet points. One bullet point is highlighted with a red rectangular box around the word "const". To the right of the monitor, there is a stylized illustration of a circuit board or a map.

DEFINING CONSTANTS

- Defining Constants - by defining a constant variable

Format:

```
const type varName = value;
```

type can be int, float, char, etc.

Content Copyright Nanyang Technological University 68

where **type** can be **int**, **float**, **char**, etc

The slide has a red header bar with the title "DEFINING CONSTANTS". Below the header, there is a list of bullet points and some explanatory text. A code snippet is shown in an orange box, and a descriptive sentence is also in an orange box. The background of the slide features a faint illustration of a computer monitor displaying code and a circuit board.

• Defining Constants - by defining a constant variable

Format:

```
const type varName = value;
```

varName is the name of the constant variable.

Content Copyright Nanyang Technological University 69

**varName** is the name of the constant variable

## DEFINING CONSTANTS

- Defining Constants - by defining a constant variable

Format:

```
const type varName = value;
```

**value** is the constant value to be assigned to the constant variable

Content Copyright Nanyang Technological University 70

**value** is the constant value to be assigned to the constant variable

## DEFINING CONSTANTS

- Defining Constants - by defining a constant variable

Format:

```
const type varName = value;
```

```
const float pi= 3.14159;
```

C provides **#define** and **const** to define symbolic names for constants. We recommend using **#define** to name constants of simple data types, and **const** to define constants that depend on another constant.

Content Copyright Nanyang Technological University 71

for example, `const float pi = 3.14159;` C provides #define and const to define symbolic names for constants

## DEFINING CONSTANTS

For example:

```
const float pi= 3.14159;
```

C provides **#define** and **const** to define symbolic names for constants. We recommend using **#define** to name constants of simple data types, and **const** to define constants that depend on another constant.

Content Copyright Nanyang Technological University 72

For example, it declares a float constant variable pi with value 3.14159. C provides **#define** and **const** to define symbolic names for constants. We recommend using **#define** to name constants of simple data types, and **const** to define constants that depend on another constant.

The slide has a red header bar with the title "DEFINING CONSTANTS". Below the header, there is a text box containing the following code:

```
#define TAX_RATE 0.12  
const double monthRate = TAX_RATE/12;
```

Below the code, there is an orange callout box containing the text: "#define is used for defining the annual tax rate". At the bottom of the slide, there is a footer bar with the text "Content Copyright Nanyang Technological University" and the number "73".

For example, in the declarations  
**#define** is used for defining the annual tax rate

The diagram shows a computer monitor with a white frame. On the screen, there is a red header bar at the top with the text "DEFINING CONSTANTS". Below this, a light gray rectangular area contains the text "For example:". In the center of the screen, there is a code block with an orange background. The code is:  

```
#define TAX_RATE 0.12  
const double monthRate = TAX_RATE/12;
```

Below the code, there is another orange rectangular callout box containing the text: "**const** is used to define the monthly tax rate which is one-twelfth of the annual tax rate." At the bottom of the monitor's screen, there is a thin white horizontal bar with the text "Content Copyright Nanyang Technological University" on the left and the number "74" on the right. The background behind the monitor is a grayscale image of a circuit board.

**While const** is used to define the monthly tax rate which is one-twelfth of the annual tax rate.

**VARIABLES**

- Programs work with data, and data is stored in memory.

Content Copyright Nanyang Technological University 75

### **Variables**

Programs work with data, and data is stored in memory.

## VARIABLES

- Programs work with data, and data is stored in memory.
- Variables are symbolic names that are used to store data in memory.

Content Copyright Nanyang Technological University

76

Variables are symbolic names that are used to store data in memory.

## VARIABLES

- Programs work with data, and data is stored in memory.
- Variables are symbolic names that are used to store data in memory.
- The content in the memory location is the current value of the variable. We use the variable name in order to retrieve the value stored at the memory location.

Content Copyright Nanyang Technological University

77

The content in the memory location is the current value of the variable. We use the variable name in order to retrieve the value stored at the memory location.

## VARIABLES

- Programs work with data, and data is stored in memory.
- Variables are symbolic names that are used to store data in memory.
- The content in the memory location is the current value of the variable. We use the variable name in order to retrieve the value stored at the memory location.
- Variables are different from constants in that the value of a variable may change during program execution, while the value of a constant remains unchanged throughout the program execution.

Content Copyright Nanyang Technological University

78

Variables are different from constants in that the value of a variable may change during program execution, while the value of a constant remains unchanged throughout the program execution.

The slide has a red header bar with the word "VARIABLES" in white capital letters. Below the header is a large grey rounded rectangle containing a bulleted list. At the bottom of the slide, there are two rectangular boxes: an orange one on the left labeled "OLD VALUE" and a yellow one on the right labeled "NEW VALUE". The slide is set against a background of a spiral notebook.

• When a new value is assigned to the variable, the old value is replaced with the new value.

OLD VALUE      NEW VALUE

Content Copyright Nanyang Technological University 79

When a new value is assigned to the variable, the old value is replaced with the new value.

The slide has a red header bar with the word 'VARIABLES' in white capital letters. Below the header is a large grey rounded rectangle containing a bulleted list. At the bottom of the slide, there are two rectangular boxes: an orange one on the left labeled 'OLD VALUE' and a yellow one on the right labeled 'NEW VALUE'. The slide is set against a background of a spiral notebook.

- When a new value is assigned to the variable, the old value is replaced with the new value.

Content Copyright Nanyang Technological University 80

When a new value is assigned to the variable, the old value is replaced with the new value.

## VARIABLES

- The name of a variable is made up of a sequence of alphanumeric characters and the underscore '\_'.

Content Copyright Nanyang Technological University

81

The name of a variable is made up of a sequence of alphanumeric characters and the underscore.

## VARIABLES

- The name of a variable is made up of a sequence of alphanumeric characters and the underscore '\_'.
  - The first character must be a letter or an underscore.

Content Copyright Nanyang Technological University

82

The first character must be a letter or an underscore.

## VARIABLES

- The name of a variable is made up of a sequence of alphanumeric characters and the underscore '\_'.
  - The first character must be a letter or an underscore.
  - There is a system-dependent limit to the maximum number of characters that can be used.

Content Copyright Nanyang Technological University

83

There is a system-dependent limit to the maximum number of characters that can be used.

## VARIABLES

- The name of a variable is made up of a sequence of alphanumeric characters and the underscore '\_'.
  - The first character must be a letter or an underscore.
  - There is a system-dependent limit to the maximum number of characters that can be used.
  - The ANSI C standard limit is 31.

Content Copyright Nanyang Technological University

84

The ANSI C standard limit is 31.

## VARIABLES

- Variable name in C is also case sensitive.

Variable name in C is also case sensitive.

**VARIABLES**

- Variable name in C is also case sensitive.
- The variable names **count** and **COUNT** refer to two different variables.

Content Copyright Nanyang Technological University 86

The variable names **count WRITTEN IN LOWER CASE**

The slide has a red header bar with the word 'VARIABLES' in white. Below the header is a list of bullet points:

- Variable name in C is also case sensitive.
- The variable names **count** and **COUNT** refer to two different variables.

Two yellow rounded rectangular boxes are positioned below the list. An arrow points from the left box to the word 'count'. Another arrow points from the right box to the word 'COUNT'.

Content Copyright Nanyang Technological University 87

**COUNT written in uppercase** refer to two different variables.

## VARIABLES

- Variable name in C is also case sensitive.
- The variable names **count** and **COUNT** refer to two different variables.
- However, it is conventional to use lowercase letters to name variables.

Content Copyright Nanyang Technological University

88

However, it is conventional to use lowercase letters to name variables.

## VARIABLES

- Variable name in C is also case sensitive.
- The variable names **count** and **COUNT** refer to two different variables.
- However, it is conventional to use lowercase letters to name variables.
- In addition, meaningful names make programs more readable.

Content Copyright Nanyang Technological University

89

In addition, meaningful names make programs more readable.

## VARIABLES

- Variable name in C is also case sensitive.
- The variable names **count** and **COUNT** refer to two different variables.
- However, it is conventional to use lowercase letters to name variables.
- In addition, meaningful names make programs more readable.
- For example, '**tax**' will make a program easier to understand than '**t**'.

Content Copyright Nanyang Technological University

90

For example, 'tax' will make a program easier to understand than 't'.

## VARIABLES

- Variable name in C is also case sensitive.
- The variable names **count** and **COUNT** refer to two different variables.
- However, it is conventional to use lowercase letters to name variables.
- In addition, meaningful names make programs more readable.
- For example, '**tax**' will make a program easier to understand than '**t**'.
- A variable name cannot be any of the *keywords* in C. Keywords have special meanings to C compiler.

Content Copyright Nanyang Technological University

91

A variable name cannot be any of the *keywords* in C. ()() Keywords have special meanings to C compiler.

## VARIABLES

The following C **keywords** are reserved and cannot be used as variable names:

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
struct	switch	typedef	union	sizeof	static
volatile	while	unsigned	void		

Content Copyright Nanyang Technological University 92

The following C **keywords** are reserved and cannot be used as variable names:

## VARIABLES

- Variables are declared by declaration statements.

Each variable has a type. The basic C data types are integer, floating point and character. Variables are declared by *declaration statements*.

## VARIABLES

- Variables are declared by declaration statements.

## VARIABLES

- Variables are declared by declaration statements.
- A declaration can be done with or without initialization.

A declaration can be done with or without initialization.

## VARIABLES

- Variables are declared by declaration statements.
- A declaration can be done with or without initialization.
- A declaration statement without initialization has the format:

Content Copyright Nanyang Technological University

96

A declaration statement without initialization has the format

## VARIABLES

- Variables are declared by declaration statements.
- A declaration can be done with or without initialization.
- A declaration statement without initialization has the format:

**data\_type var\_name[, var\_name];**

Content Copyright Nanyang Technological University 97

As shown

## VARIABLES

- Variables are declared by declaration statements.
- A declaration can be done with or without initialization.
- A declaration statement without initialization has the format:

**data\_type var\_name[, var\_name];**

**data\_type** can be int, char, float, double, etc.

Content Copyright Nanyang Technological University 98

where **data\_type** can be **int, char, float, double**, etc.

## VARIABLES

- Variables are declared by declaration statements.
- A declaration can be done with or without initialization.
- A declaration statement without initialization has the format:

`data_type var_name[, var_name];`

**var\_name** is the name of the variable

Content Copyright Nanyang Technological University 99

**var\_name** is the name of the variable.

## VARIABLES

- Variables are declared by declaration statements.
- A declaration can be done with or without initialization.
- A declaration statement without initialization has the format:

**data\_type var\_name[, var\_name];**

[..] may be repeated zero or more times but  
need to be separated by commas.

[..] may be repeated zero or more times but need to be separated by commas.

## VARIABLES

Declare your variables at the **beginning** of your program.

Examples of variable initialisations:

```
int count = 20;  
float temperature, result;
```

Content Copyright Nanyang Technological University 101

During program execution, memory storage of suitable size is assigned for each variable. A variable must be declared first before it is used.

## VARIABLES

Declare your variables at the **beginning** of your program.

Examples of variable initialisations:

```
int count = 20;  
float temperature, result;
```

Content Copyright Nanyang Technological University 102

The memory location is used to store the current value of the variable. Variable names can then be used to retrieve the value stored in the memory location. Initialization can also be done as part of a declaration. This means that a variable is given a starting value when it is declared.

## VARIABLES

```
int main()
{
    float tax, salary;
    int numOfChildren = 2, numOfParents = 2;
    char maritalStatus = 'M';
    ...
    return 0;
}
```

The declaration statements for **numOfChildren**, **numOfParents** and **maritalStatus** declare and initialize the variables in one step.

Content Copyright Nanyang Technological University 103

In the following program,  
The declaration statements for **numOfChildren**, **numOfParents** and **maritalStatus** declare and initialize the variables in one step.

## VARIABLES

```
int main()
{
    float tax, salary;
    int numOfChildren = 2, numOfParents = 2;
    char maritalStatus = 'M';
    ...
    return 0;
}
```

It is a very useful feature when developing complex programs.

Content Copyright Nanyang Technological University 104

It is a very useful feature when developing complex programs.

## VARIABLES

```
int main()
{
    float tax, salary;
    int numOfChildren , numOfParents = 2;
    char maritalStatus = 'M';
    ...
    return 0;
}
```

The declaration statement above only initializes **numOfParents** to 2.  
2. **numOfChildren** is not initialized.

Content Copyright Nanyang Technological University 105

The declaration statement shown here only initializes **numOfParents** to 2. **numOfChildren** is not initialized.

## VARIABLES

```
int main()
{
    float tax, salary;
    int numOfChildren , numOfParents = 2;
    char maritalStatus = 'M';
    ...
    return 0;
}
```

To improve the readability of the program, it is better to declare initialized and uninitialized variables in separate declaration statements.

Content Copyright Nanyang Technological University 106

To improve the readability of the program, it is better to declare initialized and uninitialized variables in separate declaration statements.

## VARIABLES

```
int main()
{
    float tax, salary;
    int tax
    int numOfChildren , numOfParents = 2;
    char maritalStatus = 'M';
    ...
    return 0;
}
```

Also note that a variable is not allowed to be declared twice.

Content Copyright Nanyang Technological University 107

Also note that a variable is not allowed to be declared twice.

## VARIABLES

```
int main()
{
    float tax, salary;
    int tax
    int numOfChildren , numOfParents = 2;
    char maritalStatus = 'M';
    ...
    return 0;
}
```

The declaration statements **float tax, salary;** **int tax;** will cause the compiler to issue an error message.

Content Copyright Nanyang Technological University 108

The declaration statements **float tax, salary;** **int tax;** will cause the compiler to issue an error message.



## Operators

## OPERATORS

- An operator is a symbol that causes some operations to be performed on one or more variables (or data values).

### Operators

An operator is a symbol that causes some operations to be performed on one or more variables (or data values).

## OPERATORS

- An operator is a symbol that causes some operations to be performed on one or more variables (or data values).
- Operators in C are mainly classified into

Content Copyright Nanyang Technological University

111

### Operators

Operators in C are mainly classified into

## OPERATORS

- An operator is a symbol that causes some operations to be performed on one or more variables (or data values).
- Operators in C are mainly classified into
  1. Fundamental arithmetic operators

fundamental arithmetic operators

## OPERATORS

- An operator is a symbol that causes some operations to be performed on one or more variables (or data values).
- Operators in C are mainly classified into
  1. Fundamental arithmetic operators
  2. Increment/decrement operators

Content Copyright Nanyang Technological University

113

increment/decrement operators

## OPERATORS

- An operator is a symbol that causes some operations to be performed on one or more variables (or data values).
- Operators in C are mainly classified into
  1. Fundamental arithmetic operators
  2. Increment/decrement operators
  3. Assignment operators

Content Copyright Nanyang Technological University

114

assignment operators

## OPERATORS

- An operator is a symbol that causes some operations to be performed on one or more variables (or data values).
- Operators in C are mainly classified into
  1. Fundamental arithmetic operators
  2. Increment/decrement operators
  3. Assignment operators
  4. Relational operators

Content Copyright Nanyang Technological University

115

relational operatorS

## OPERATORS

- An operator is a symbol that causes some operations to be performed on one or more variables (or data values).
- Operators in C are mainly classified into
  1. Fundamental arithmetic operators
  2. Increment/decrement operators
  3. Assignment operators
  4. Relational operators
  5. Conditional operators

Content Copyright Nanyang Technological University

116

and conditional operators. Relational and conditional operators will be discussed in the next lecture.

## FUNDAMENTAL ARITHMETIC OPERATORS

- Unary operators
- Binary operators

Content Copyright Nanyang Technological University

117

### Fundamental Arithmetic Operators

unary operators and binary operators ARE TWO TYPES OF FUNDAMENTAL ARITHMETIC OPERATORS

## FUNDAMENTAL ARITHMETIC OPERATORS

- Unary operators:  $+$ ,  $-$ 
  - E.g.  $+31$ ,  $-5$

Content Copyright Nanyang Technological University

118

Unary operators are positive, negative e.g. POSITIVE 31, NEGATIVE 5. They can be used to change the sign of a value

## FUNDAMENTAL ARITHMETIC OPERATORS

- Unary operators:  $+$ ,  $-$ 
  - E.g.  $+31$ ,  $-5$
- Binary operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$

Content Copyright Nanyang Technological University

119

Binary operators are addition ( $+$ ), subtraction ( $-$ ), multiplication ( $*$ ), division ( $/$ ) and modulus ( $\%$ ).

## FUNDAMENTAL ARITHMETIC OPERATORS

- Unary operators:  $+$ ,  $-$ 
  - E.g.  $+31$ ,  $-5$
- Binary operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ 
  - E.g.  $6.6/2.0=3.3$ ;  $7/3 = 2$ ;

Content Copyright Nanyang Technological University

120

### Fundamental Arithmetic Operators

Note that the division of floating point numbers (or one floating point number with an integer) returns a floating point number.

## FUNDAMENTAL ARITHMETIC OPERATORS

- Unary operators:  $+$ ,  $-$ 
  - E.g.  $+31$ ,  $-5$
- Binary operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ 
  - E.g.  $6.6/2.0=3.3$  ;  $7/3 = 2$ ;

Content Copyright Nanyang Technological University

121

However, the division operator ( $/$ ) for integers returns integer results.

## FUNDAMENTAL ARITHMETIC OPERATORS

- Unary operators:  $+$ ,  $-$ 
  - E.g.  $+31$ ,  $-5$
- Binary operators:  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$ 
  - E.g.  $6.6/2.0=3.3$  ;  $7/3 = 2$ ;  $7\%3 = 1$ ;

Content Copyright Nanyang Technological University

122

The modulus operator (%) returns the remainder of an integer division.

## FUNDAMENTAL ARITHMETIC OPERATORS

- Unary operators:  $+$ ,  $-$ 
  - E.g.  $+31, -5$
- Binary operators:  $+, -, *, /, \%$ 
  - E.g.  $6.6/2.0=3.3 ; 7/3 = 2; 7\%3 = 1;$

Examples:

$$\begin{aligned}15.0/4 &= 3.75 \\15/4.0 &= 3.75 \\15/4 &= 3 \\18/4 &= 4 \\15\%4 &= 3 \\18\%4 &= 2 \\-18\%4 &= 2\end{aligned}$$

Content Copyright Nanyang Technological University

123

Few examples are shown here.

## FUNDAMENTAL ARITHMETIC OPERATORS

- It is also possible to perform arithmetic operations on character variables.

Content Copyright Nanyang Technological University

124

It is also possible to perform arithmetic operations on character variables.

## FUNDAMENTAL ARITHMETIC OPERATORS

- It is also possible to perform arithmetic operations on character variables.

```
char chr1, chr2;  
chr1 = 'A'; /* assign 'A' to the variable chr1 */  
chr2 = chr1 + 32; /* add 32 to the variable chr1 */
```

Few examples are shown here.

## FUNDAMENTAL ARITHMETIC OPERATORS

- It is also possible to perform arithmetic operations on character variables.

```
char chr1, chr2;
chr1 = 'A'; /* assign 'A' to the variable chr1 */
chr2 = chr1 + 32; /* add 32 to the variable chr1 */
```

- The last statement essentially converts the uppercase letter into the corresponding lowercase character. This is done by adding 32 to the ASCII code for the corresponding uppercase character.

The last statement essentially converts the uppercase letter into the corresponding lowercase character. This is done by adding 32 to the ASCII code for the corresponding uppercase character.

## FUNDAMENTAL ARITHMETIC OPERATORS

- It is also possible to perform arithmetic operations on character variables.

```
char chr1, chr2;
chr1 = 'A'; /* assign 'A' to the variable chr1 */
chr2 = chr1 + 32; /* add 32 to the variable chr1 */
```

- The last statement essentially converts the uppercase letter into the corresponding lowercase character. This is done by adding 32 to the ASCII code for the corresponding uppercase character.
- The result of the assignment statement is the character constant 'a' to be assigned to the variable chr2.

Content Copyright Nanyang Technological University

127

The result of the assignment statement is the character constant '**eh**' to be assigned to the variable **chr2**. This is a very useful technique in converting an uppercase letter to its lowercase counterpart, or vice versa.

## ASSIGNMENT OPERATORS

Assignment operators: =

- E.g. float amount = 25.50;

Content Copyright Nanyang Technological University

128

### Assignment Operators

An assignment operator (=) is used to assign a value to a variable. It is the basic building block of a program.

## ASSIGNMENT OPERATORS

Assignment operators: =

- E.g. float amount = 25.50;

`var_name = expression;`

Content Copyright Nanyang Technological University 129

The value of a variable may be changed by the assignment statement in the following format:

**var\_name = expression;**

where **var\_name** is a variable name, and the **expression** (on the right-hand side) provides a value that is assigned to the variable **var\_name** on the left-hand side.

## ASSIGNMENT OPERATORS

Assignment operators: =

- E.g. float amount = 25.50;

var\_name = expression;

- An expression may be a constant (e.g. **2**, '**h**', **-6.7**), a variable (e.g. **tax**, **salary**), or a combination of operators and operands (e.g. **a+b/c**).

Content Copyright Nanyang Technological University 130

An expression may be a constant (e.g. **2**, '**h**', **-6.7**), a variable (e.g. **tax**, **salary**), or a combination of operators and operands (e.g. **a+b/c**).

## ASSIGNMENT OPERATORS

Assignment operators: =

- E.g. float amount = 25.50;

```
var_name = expression;
```

- An expression may be a constant (e.g. **2**, '**h**', **-6.7**), a variable (e.g. **tax**, **salary**), or a combination of operators and operands (e.g. **a+b/c**).
- Notice that the left-hand side must be a variable name and cannot be a constant or expression with operators.

Content Copyright Nanyang Technological University 131

Notice that the left-hand side must be a variable name and cannot be a constant or expression with operators.

## ASSIGNMENT OPERATORS

Assignment operators: =

- E.g. float amount = 25.50;

var\_name = expression;

- An expression may be a constant (e.g. **2**, '**h**', **-6.7**), a variable (e.g. **tax**, **salary**), or a combination of operators and operands (e.g. **a+b/c**).
- Notice that the left-hand side must be a variable name and cannot be a constant or expression with operators.
- The expression on the right-hand side can be a constant, a variable or an arithmetic expression.

Content Copyright Nanyang Technological University 132

The expression on the right-hand side can be a constant, a variable or an arithmetic expression.

## ASSIGNMENT OPERATORS

Consider the following assignment statement:

**num = 3;**

- The memory location of the variable **num** is used to store the value 3.
- The *value* of the variable refers to the value or content of the memory location, i.e. the actual value of the variable.



Content Copyright Nanyang Technological University

133

Consider the following assignment statement:

**num = 3;**

When a variable, e.g. **num**, is declared, the compiler allocates a memory location for storing the value of that variable. The memory location of the variable **num** is used to store the value 3. The *value* of the variable refers to the value or content of the memory location, i.e. the actual value of the variable.

## ASSIGNMENT OPERATORS

In the following assignment statement:

**count = num;**

the value of **num** is assigned to the variable **count**.



Content Copyright Nanyang Technological University

134

In the following assignment statement:

**count = num;**

the value of **num** is assigned to the variable **count**.

## ASSIGNMENT OPERATORS

The statements:

**3 = num;**

**num + count = 3;**

are *illegal* as the left-hand side is not a memory location.

Content Copyright Nanyang Technological University

135

The statements

**3 = num;**

**num + count = 3;**

are *illegal* as the left-hand side is not a memory location

## ASSIGNMENT OPERATORS

In the first statement,

**3 = num;**

the left-hand side is a constant.

Content Copyright Nanyang Technological University

136

In the first statement, the left-hand side is a constant

## ASSIGNMENT OPERATORS

In the second statement,

**num + count = 3;**

the left-hand side is an arithmetic expression, which does not represent a memory location.

Content Copyright Nanyang Technological University

137

and in the second statement, the left-hand side is an arithmetic expression, which does not represent a memory location.

## ARITHMETIC ASSIGNMENT OPERATORS

Arithmetic assignment operators: `+=, -=, *=, /=, %=`

- E.g. `a += 5;`

```
var_name op = expression;
```

where the arithmetic assignment operators (**op=**) are `+=, -=, *=, /=` and `%=`.

## ARITHMETIC ASSIGNMENT OPERATORS

- Arithmetic assignment operators: `+ =`, `- =`, `* =`, `/ =`, `% =`
  - E.g. `a += 5;`

```
var_name op = expression;
```

where the arithmetic assignment operators (**op=**) are `+ =`, `- =`, `* =`, `/ =` and `% =`.

Content Copyright Nanyang Technological University

139

In addition to the assignment operators, C also has arithmetic assignment operators, which combine an arithmetic operator with the assignment operator. Arithmetic assignment operator statement has the format as shown:

where the arithmetic assignment operators are `+ =`, minus equal, multiply equal, divide equal and modulus equal

## ARITHMETIC ASSIGNMENT OPERATORS

- Arithmetic assignment operators: `+ =`, `- =`, `* =`, `/ =`, `% =`
  - E.g. `a += 5;`

```
var_name op = expression;
```

where the arithmetic assignment operators (**op=**) are `+ =`, `- =`, `* =`, `/ =` and `% =`.

- This is equivalent to:

```
var_name = var_name op expression;
```

- The arithmetic assignment operator first performs the arithmetic operation specified by the arithmetic operator, and assigns the resulting value to the variable.

Content Copyright Nanyang Technological University

140

The arithmetic assignment operator first performs the arithmetic operation specified by the arithmetic operator, and assigns the resulting value to the variable.

## INCREMENT OPERATORS

- The increment operator (`++`) increases a variable by 1.
- It can be used in two modes: *prefix* and *postfix*.

Content Copyright Nanyang Technological University

141

### Increment/decrement Operators

The increment operator (`++`) increases a variable by 1. It can be used in two modes: *prefix* and *postfix*.

**INCREMENT OPERATORS**

In **prefix mode**:

**`++var_name;`**

`var_name` is incremented by 1 and the value of the expression is the updated value of `var_name`.

Content Copyright Nanyang Technological University

142

The format of the prefix mode is

**`++var_name;`**

where **`var_name`** is incremented by 1 and the value of the expression is the updated value of **`var_name`**.

## INCREMENT OPERATORS

In **prefix mode**:

`++var_name;`

var\_name is incremented by 1 and the value of the expression is the updated value of var\_name.

In **postfix mode**:

`var_name++;`

The value of the expression is the current value of var\_name and then var\_name is incremented by 1.

Content Copyright Nanyang Technological University

143

The format of the postfix mode is

**`var_name++;`**

where the value of the expression is the current value of **var\_name** and then **var\_name** is incremented by 1.

Notice that one important difference between the prefix and postfix modes is on the time when that operation is performed. In the prefix mode, the variable is incremented before any operation with it, while in the postfix mode, the variable is incremented after any operation with it.

## INCREMENT OPERATORS

**Code**

**Output**

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
    num++; // ++num; i.e., num =num+1;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num++ is %d\n", num++);
    printf("value of num is %d\n", num);
    printf("value of ++num is %d\n", ++num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

Content Copyright Nanyang Technological University

144

In the example program, it shows some examples on the use of increment operators.

## INCREMENT OPERATORS

Code

Output

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
    num++; // ++num; i.e., num =num+1;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num++ is %d\n", num++);
    printf("value of num is %d\n", num);
    printf("value of ++num is %d\n", ++num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

Content Copyright Nanyang Technological University

145

The initial value of the variable **num** is 4.

The slide has a red header bar with the title "INCREMENT OPERATORS". Below the header, there are two main sections: "Code" and "Output".

**Code:**

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
    num++; // ++num; i.e., num =num+1;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num++ is %d\n", num++);
    printf("value of num is %d\n", num);
    printf("value of ++num is %d\n", ++num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

**Output:**

value of num is 4

Content Copyright Nanyang Technological University 146

The first **printf()** statement prints the value of 4 for **num**.

## INCREMENT OPERATORS

Code	Output
<pre>#include &lt;stdio.h&gt; int main() {     int num = 4;     printf("value of num is %d\n", num);     num++; // ++num; i.e., num =num+1;     printf("value of num is %d\n", num);     num = 4;     printf("value of num++ is %d\n", num++);     printf("value of num is %d\n", num);     printf("value of ++num is %d\n", ++num);     printf("value of num is %d\n\n", num);     return 0; }</pre>	<p>value of num is 4 value of num is 5</p>

Content Copyright Nanyang Technological University

147

The second **printf()** statement prints the value of 5 after increment. The variable **num** is assigned with the value 4 again.

## INCREMENT OPERATORS

Code	Output
<pre>#include &lt;stdio.h&gt; int main() {     int num = 4;     printf("value of num is %d\n", num);     num++; // ++num; i.e., num =num+1;     printf("value of num is %d\n", num);     num = 4;     printf("value of num++ is %d\n", num++);     printf("value of num is %d\n", num);     printf("value of ++num is %d\n", ++num);     printf("value of num is %d\n\n", num);     return 0; }</pre>	<pre>value of num is 4 value of num is 5 value of num++ is 4</pre>

Content Copyright Nanyang Technological University

148

As the third **printf()** statement uses the postfix mode, the value of **num** is incremented by 1 after the printing has taken place. Therefore, the third **printf()** statement prints the value of 4 for **num**.

## INCREMENT OPERATORS

**Code**

**Output**

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
    num++; // ++num; i.e., num =num+1;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num++ is %d\n", num++);
    printf("value of num is %d\n", num);
    printf("value of ++num is %d\n", ++num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

```
value of num is 4
value of num is 5
value of num++ is 4
value of num is 5
```

Content Copyright Nanyang Technological University

149

The fourth statement then prints the value of 5.

## INCREMENT OPERATORS

Code	Output
<pre>#include &lt;stdio.h&gt; int main() {     int num = 4;     printf("value of num is %d\n", num);     num++; // ++num; i.e., num =num+1;     printf("value of num is %d\n", num);     num = 4;     printf("value of num++ is %d\n", num++);     printf("value of num is %d\n", num);     printf("value of ++num is %d\n", ++num);     printf("value of num is %d\n\n", num);     return 0; }</pre>	<pre>value of num is 4 value of num is 5 value of num++ is 4 value of num is 5 value of ++num is 6</pre>

Content Copyright Nanyang Technological University

150

The fifth **printf()** statement uses the prefix mode. The value of **num** is incremented before the printing is taken place. Thus, it prints the value 6.

## INCREMENT OPERATORS

Code	Output
<pre>#include &lt;stdio.h&gt; int main() {     int num = 4;     printf("value of num is %d\n", num);     num++; // ++num; i.e., num =num+1;     printf("value of num is %d\n", num);     num = 4;     printf("value of num++ is %d\n", num++);     printf("value of num is %d\n", num);     printf("value of ++num is %d\n", ++num);     printf("value of num is %d\n\n", num);     return 0; }</pre>	<pre>value of num is 4 value of num is 5 value of num++ is 4 value of num is 5 value of ++num is 6 value of num is 6</pre>

Content Copyright Nanyang Technological University

151

The sixth statement also prints the value 6 for **num**.

## DECREMENT OPERATORS

The way the **decrement operator** '--' works in the same way as the ++ operator, except that the variable is decremented by 1.

Content Copyright Nanyang Technological University 152

The decrement operator (--) works in a similar way as the ++ operator, except that the variable is decremented by 1.

## DECREMENT OPERATORS

**Code**

**Output**

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
num--; // same as --num;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num-- is %d\n", num--);
    printf("value of num is %d\n", num);
    printf("value of --num is %d\n", --num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

Content Copyright Nanyang Technological University

153

In the example program, it shows some examples on the use of decrement operators.

## DECREMENT OPERATORS

Code

Output

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
num--; // same as --num;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num-- is %d\n", num--);
    printf("value of num is %d\n", num);
    printf("value of --num is %d\n", --num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

Content Copyright Nanyang Technological University

154

The initial value of the variable **num** is 4.

## DECREMENT OPERATORS

Code

Output

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
    num--; // same as --num;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num-- is %d\n", num--);
    printf("value of num is %d\n", num);
    printf("value of --num is %d\n", --num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

value of num is 4

Content Copyright Nanyang Technological University

155

- . The first **printf()** statement prints the value of 4 for **num**.

## DECREMENT OPERATORS

Code	Output
<pre>#include &lt;stdio.h&gt; int main() {     int num = 4;     printf("value of num is %d\n", num);     num--; // same as --num;     printf("value of num is %d\n", num);     num = 4;     printf("value of num-- is %d\n", num--);     printf("value of num is %d\n", num);     printf("value of --num is %d\n", --num);     printf("value of num is %d\n\n", num);     return 0; }</pre>	<p>value of num is 4 value of num is 3</p>

Content Copyright Nanyang Technological University

156

The second **printf()** statement prints the value of 3 after decrement.

## DECREMENT OPERATORS

Code

Output

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
num--; // same as --num;
    printf("value of num is %d\n", num);
    num = 4;
printf("value of num-- is %d\n", num--);
    printf("value of num is %d\n", num);
    printf("value of --num is %d\n", --num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

```
value of num is 4
value of num is 3
value of num-- is 4
```

Content Copyright Nanyang Technological University

157

The variable **num** is assigned with the value 4 again.

## DECREMENT OPERATORS

**Code**

**Output**

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
    num--; // same as --num;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num-- is %d\n", num--);
    printf("value of num is %d\n", num);
    printf("value of --num is %d\n", --num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

```
value of num is 4
value of num is 3
value of num-- is 4
value of num is 3
```

Content Copyright Nanyang Technological University

158

As the third **printf()** statement uses the postfix mode, the value of **num** is decremented by 1 after the printing has taken place. Therefore, the third **printf()** statement prints the value of 3 for **num**.

## DECREMENT OPERATORS

**Code**

**Output**

```
#include <stdio.h>
int main()
{
    int num = 4;
    printf("value of num is %d\n", num);
    num--; // same as --num;
    printf("value of num is %d\n", num);
    num = 4;
    printf("value of num-- is %d\n", num--);
    printf("value of num is %d\n", num);
    printf("value of --num is %d\n", --num);
    printf("value of num is %d\n\n", num);
    return 0;
}
```

```
value of num is 4
value of num is 3
value of num-- is 4
value of num is 3
value of --num is 2
```

Content Copyright Nanyang Technological University

159

The fourth statement then prints the value of 2.

## DECREMENT OPERATORS

Code	Output
<pre>#include &lt;stdio.h&gt; int main() {     int num = 4;     printf("value of num is %d\n", num); <b>num--;</b> // same as --num;     printf("value of num is %d\n", num);     num = 4;     printf("value of num-- is %d\n", <b>num--</b>);     printf("value of num is %d\n", num);     printf("value of --num is %d\n", <b>--num</b>); <b>printf("value of num is %d\n\n",num);</b>     return 0; }</pre>	<pre>value of num is 4 value of num is 3 value of num-- is 4 value of num is 3 value of --num is 2 value of num is 2</pre>

Content Copyright Nanyang Technological University

160

The value of **num** is decremented before the printing is taken place. Thus, it prints the value 2. The sixth statement also prints the value 2 for **num**.

## ARITHMETIC ASSIGNMENT OPERATORS

The assignment statements can also involve increment/decrement operators:

The assignment statements can also involve increment/decrement operators:

## ARITHMETIC ASSIGNMENT OPERATORS

The assignment statements can also involve increment/decrement operators:

```
var_name1 = var_name2++;
var_name1 = ++var_name2;
var_name1 = var_name2--;
var_name1 = --var_name2;
```

Content Copyright Nanyang Technological University

162

as shown here.

## ARITHMETIC ASSIGNMENT OPERATORS

The assignment statements can also involve increment/decrement operators:

```
var_name1 = var_name2++;
var_name1 = ++var_name2;
var_name1 = var_name2--;
var_name1 = --var_name2;
```

```
int m, k=2;
m = k++;
```

Content Copyright Nanyang Technological University

163

For example, if we define two variables **m** and **k**

**int m, k=2;**

the statement

**m = k++;**

## ARITHMETIC ASSIGNMENT OPERATORS

The assignment statements can also involve increment/decrement operators:

```
var_name1 = var_name2++;  
var_name1 = ++var_name2;  
var_name1 = var_name2--;  
var_name1 = --var_name2;
```

```
int m, k=2;  
m = k++;
```

```
m = k;  
k = k+1;
```

Content Copyright Nanyang Technological University

164

which is the same as

**m = k;**  
**k = k + 1;**

## ARITHMETIC ASSIGNMENT OPERATORS

The assignment statements can also involve increment/decrement operators:

```
var_name1 = var_name2++;  
var_name1 = ++var_name2;  
var_name1 = var_name2--;  
var_name1 = --var_name2;
```

```
int m, k=2;  
m = k++;
```

```
m = k;  
k = k+1;
```

```
m = 2;  
k = 3;
```

Content Copyright Nanyang Technological University

165

will give **m** the value 2 and **k** the value 3 after executing the statement.

## ARITHMETIC ASSIGNMENT OPERATORS

The assignment statements can also involve increment/decrement operators:

```
var_name1 = var_name2++;  
var_name1 = ++var_name2;  
var_name1 = var_name2--;  
var_name1 = --var_name2;
```

```
int m, k=2;  
m = ++k;
```

```
k = k+1;  
m = k;
```

```
k = 3;  
m = 3;
```

Content Copyright Nanyang Technological University

166

However, the statement

**m = ++k;**

which is the same as

**k = k + 1;**

**m = k;**

will give **m** the value 3 and **k** the value 3 after executing the statement.

The slide has a red header bar with the title 'CHAINED ASSIGNMENT OPERATORS'. Below the header is a text block stating 'Chained assignments are also possible in one statement:' followed by a code example: 'var\_name1 =var\_name2=...=var\_nameN= expression;'. At the bottom left is the copyright notice 'Content Copyright Nanyang Technological University' and at the bottom right is the page number '167'.

Content Copyright Nanyang Technological University 167

### **Chained Assignments**

Chained assignments are also possible in one statement as shown here.

## CHAINED ASSIGNMENT OPERATORS

Chained assignments are also possible in one statement:

```
var_name1 = var_name2=...=var_nameN= expression;
```

This is equivalent to

```
var_nameN = expression;  
....  
var_name2 = expression;  
var_name1 = expression;
```

Content Copyright Nanyang Technological University

168

This is equivalent to

```
var_name2 = expression;  
var_name1 = expression;
```

**DATA TYPE CONVERSION**

- Data type conversion is the conversion of one data type into another type.

Content Copyright Nanyang Technological University 169

### Data Type Conversion

Data type conversion is the conversion of one data type into another type.

## DATA TYPE CONVERSION

- Data type conversion is the conversion of one data type into another type.
- Arithmetic operations require **two numbers** in an expression/assignment are of the **same type**.

Content Copyright Nanyang Technological University 170

Data type conversion is the conversion of one data type into another type. It is needed when more than one type of data objects appear in an expression or assignment.

## DATA TYPE CONVERSION

- Data type conversion is the conversion of one data type into another type.
- Arithmetic operations require **two numbers** in an expression/assignment are of the **same type**.
- For example, the statement **a = 5 + 2.68;** adds two numbers with different data types, i.e. *integer* and *floating point*. However, the addition operation can only be done if these two numbers are of the same type.
- Type conversion will take place when two operands of an expression are of different data types.

Content Copyright Nanyang Technological University 171

[For example, the statement

$a = 5 + 2.68;$

adds two numbers with different data types, i.e. integer and floating point.

However, the addition operation can only be done if these two numbers are of the same type.

Type conversion will take place when two operands of an expression are of different data types

**DATA TYPE CONVERSION**

Three kinds of conversion are available:

1. Explicit conversion
2. Arithmetic conversion
3. Assignment conversion

Content Copyright Nanyang Technological University 172

Three kinds of conversion can be performed: (1) explicit conversion; (2) arithmetic conversion; and (3) assignment conversion.

## EXPLICIT CONVERSION

- It uses the type cast operator, (**type**), e.g. (**int**), (**float**), (**double**), etc. to force the compiler to convert the value of the operand into the type specified by the operator.

In *explicit conversion*, it uses the type cast operator, (**type**), e.g. (**int**), (**float**), (**double**), etc. to force the compiler to convert the value of the operand into the type specified by the operator.

## EXPLICIT CONVERSION

- It uses the type cast operator, (**type**), e.g. (**int**), (**float**), (**double**), etc. to force the compiler to convert the value of the operand into the type specified by the operator.

```
int num=3;  
float result;  
result = (float)num;
```

For example, the statement is shown here.

## EXPLICIT CONVERSION

- It uses the type cast operator, (**type**), e.g. (**int**), (**float**), (**double**), etc. to force the compiler to convert the value of the operand into the type specified by the operator.

```
int num;  
float result;  
result = (float)num;
```

- Explicitly converts the value of **num** into data type **float** and assign the value to the variable **result** of data type **float**.  
e.g. (int)2.7 + (int)3.5

Explicitly converts the value of **num** into data type **float** and assign the value to the variable **result** of data type **float**.

## ARITHMETIC CONVERSION

- It performs automatic conversion in any operation that involves operands of two different types.

In *arithmetic conversion*, it performs automatic conversion in any operation that involves operands of two different types.

## ARITHMETIC CONVERSION

- It performs automatic conversion in any operation that involves operands of two different types.
- It converts the operands to the type of the **higher ranking** of the two. This process is also called *promotion*.

Content Copyright Nanyang Technological University

177

It converts the operands to the type of the higher ranking of the two. This process is also called *promotion*.

## ARITHMETIC CONVERSION

- It performs automatic conversion in any operation that involves operands of two different types.
- It converts the operands to the type of the **higher ranking** of the two. This process is also called *promotion*.
- The ranking of types is based on the amount of memory storage the value needs.

The diagram illustrates the ranking of C++ data types based on memory storage requirements. A vertical arrow points downwards, indicating the progression from **High** to **Low** memory needs. The types listed are: long double, double, float, long, and Int. The **High** end of the scale is at the top, and the **Low** end is at the bottom.

long double
double
float
long
Int

Content Copyright Nanyang Technological University 178

The ranking of types is based on the amount of memory storage the value needs.  
The ranking from high to low is as indicated by arrows.

## ARITHMETIC CONVERSION

```
float ans1, ans2;  
ans1 = 1.23 + 5/4; /* first statement */  
ans2 = 1.23 + 5.0/4; /* second statement */
```

5/4 =1

In the first statement, the expression **5/4** will be evaluated using integer division. It gives the result of 1.

Content Copyright Nanyang Technological University 179

In the first statement, the expression **5/4** will be evaluated using integer division.

## ARITHMETIC CONVERSION

```
float ans1, ans2;
ans1 = 1.23 + 5/4; /* first statement */
ans2 = 1.23 + 5.0/4; /* second statement */
```

1.23+1.0=2.23

This result is then converted to 1.0 and added to the floating point constant 1.23 to give the final result of 2.23.

Content Copyright Nanyang Technological University 180

This result is then converted to 1.0 and added to the floating point constant 1.23 to give the final result of 2.23.

## ARITHMETIC CONVERSION

```
float ans1, ans2;
ans1 = 1.23 + 5/4; /* first statement */
ans2 = 1.23 + 5.0/4; /* second statement */
```

ans1=2.23

This value is then assigned to the variable **ans1**.

Content Copyright Nanyang Technological University 181

This value is then assigned to the variable **ans1**.

## ARITHMETIC CONVERSION

```
float ans1, ans2;  
ans1 = 1.23 + 5/4; /* first statement */  
ans2 = 1.23 + 5.0/4; /* second statement */
```

$$5.0/4=5.0/4.0=1.25$$

In the second statement, the expression **5.0/4** will be evaluated using floating point arithmetic. The value 4 in the expression will be converted to 4.0 before evaluating the expression to get 1.25.

In the second statement, the expression **5.0/4** will be evaluated using floating point arithmetic. The value 4 in the expression will be converted to 4.0 before evaluating the expression to get 1.25.

## ARITHMETIC CONVERSION

```
float ans1, ans2;
ans1 = 1.23 + 5/4; /* first statement */
ans2 = 1.23 + 5.0/4; /* second statement */
```

1.23+1.25 =2.48

The result of 1.25 is then added to 1.23 to give the final result of 2.48.

Content Copyright Nanyang Technological University 183

The result of 1.25 is then added to 1.23 to give the final result of 2.48.

## ARITHMETIC CONVERSION

```
float ans1, ans2;
ans1 = 1.23 + 5/4; /* first statement */
ans2 = 1.23 + 5.0/4; /* second statement */
```

ans2=2.48

The result is then assigned to the variable **ans2**.

Content Copyright Nanyang Technological University 184

- The result is then assigned to the variable **ans2**.

## ASSIGNMENT CONVERSION

- Converts the type of the result of computing the expression to that of the type of the left hand side if they are different.  
–e.g. int b; b = **2.7 + 3.5**; // 6.2 to 6 then to b

In *assignment conversion*, it converts the type of the result of computing the expression to that of the type of the left-hand side if they are different.

### ASSIGNMENT CONVERSION

- Converts the type of the result of computing the expression to that of the type of the left hand side if they are different.  
–e.g. int b; b = **2.7 + 3.5**; // 6.2 to 6 then to b
- If the variable on the left-hand side of the assignment statement has a higher rank or same rank as the expression, then there is no loss of precision. Otherwise, there could be a loss of accuracy.

Content Copyright Nanyang Technological University

186

if the variable on the left-hand side of the assignment statement has a higher rank or same rank as the expression, then there is no loss of precision. Otherwise, there could be a loss of accuracy.

## ASSIGNMENT CONVERSION

```
int i;
float x=2.5, y=5.3;
i = x + y;
```

- If an expression has a floating point result, which is assigned to an integer variable, the fractional part of the result will be lost.

Content Copyright Nanyang Technological University 187

For example, if an expression has a floating point result, which is assigned to an integer variable, the fractional part of the result will be lost.

## ASSIGNMENT CONVERSION

```
int i;
float x=2.5, y=5.3;
i = x + y;
```

- If an expression has a floating point result, which is assigned to an integer variable, the fractional part of the result will be lost.  
2.5+5.3=7.8      Output is 7
- This is because the lower ranking type may not have enough memory storage to store the value of higher ranking type.

Content Copyright Nanyang Technological University      188

Though the value computed is 7.8, the statement will give **i** the value 7. This is because the lower ranking type may not have enough memory storage to store the value of higher ranking type.

**PIT FALLS: DATA TYPE CONVERSION**

Loss of precision: e.g. from **float** to **int**, the fractional part will be lost.

Content Copyright Nanyang Technological University 189

Possible pit-falls about type conversion is the Loss of precision, for example, when it is converted from **float** to **int**, the fractional part will be lost.

## MATHEMATICAL LIBRARIES - #include <math.h>

- All C compilers provide a set of mathematical functions in the standard library.

Content Copyright Nanyang Technological University

190

### Mathematical Libraries

All C compilers provide a set of mathematical functions in the standard library.

## MATHEMATICAL LIBRARIES - #include <math.h>

- All C compilers provide a set of mathematical functions in the standard library.
- Some of the common mathematical functions include square root **sqrt()**, power **pow()**, and absolute values **abs()** and **fabs()**.

Content Copyright Nanyang Technological University

191

Some of the common mathematical functions include square root **sqrt()**, power **pow()**, and absolute values **abs()** and **fabs()**.

## MATHEMATICAL LIBRARIES - #include <math.h>

Function	Argument Type	Description	Result Type
ceil(x)	double	Return the smallest double larger than or equal to x that can be represented as an int.	double
floor(x)	double	Return the largest double smaller than or equal to x that can be represented as an int.	double
abs(x)	int	Return the absolute value of x, where x is an int.	int
fabs(x)	double	Return the absolute value of x, where x is a floating point number.	double
<b>sqrt(x)</b>	<b>double</b>	<b>Return the square root of x, where x &gt;= 0.</b>	<b>double</b>
pow(x,y)	double x, double y	Return x to the y power, $x^y$ .	double
cos(x)	double	Return the cosine of x, where x is in radians.	double
sin(x)	double	Return the sine of x, where x is in radians.	double
tan(x)	double	Return the tangent of x, where x is in radians.	double
exp(x)	double	Return the exponential of x with the base e, where e is 2.718282.	double
log(x)	double	Return the natural logarithm of x.	double
log10(x)	double	Return the base 10 logarithm of x.	double

Content Copyright Nanyang Technological University

192

### Mathematical Libraries

The **sqrt()** function computes the square root of a value of type **double**. It returns the result of type **double**. The statement **y = sqrt(x);** computes the square root of the value stored in the variable **x**, returns and stores the result in the variable **y**.

## MATHEMATICAL LIBRARIES - #include <math.h>

Function	Argument Type	Description	Result Type
ceil(x)	double	Return the smallest double larger than or equal to x that can be represented as an int.	double
floor(x)	double	Return the largest double smaller than or equal to x that can be represented as an int.	double
<b>abs(x)</b>	<b>int</b>	Return the absolute value of x, where x is an int.	<b>int</b>
<b>fabs(x)</b>	<b>double</b>	Return the absolute value of x, where x is a floating point number.	<b>double</b>
sqrt(x)	double	Return the square root of x, where x >= 0.	double
pow(x,y)	double x, double y	Return x to the y power, $x^y$ .	double
cos(x)	double	Return the cosine of x, where x is in radians.	double
sin(x)	double	Return the sine of x, where x is in radians.	double
tan(x)	double	Return the tangent of x, where x is in radians.	double
exp(x)	double	Return the exponential of x with the base e, where e is 2.718282.	double
log(x)	double	Return the natural logarithm of x.	double
log10(x)	double	Return the base 10 logarithm of x.	double

Content Copyright Nanyang Technological University

193

The functions **abs()** and **fabs()** give the absolute values of an integer and a floating point number respectively.

## MATHEMATICAL LIBRARIES - #include <math.h>

Function	Argument Type	Description	Result Type
ceil(x)	double	Return the smallest double larger than or equal to x that can be represented as an int.	double
floor(x)	double	Return the largest double smaller than or equal to x that can be represented as an int.	double
abs(x)	int	Return the absolute value of x, where x is an int.	int
fabs(x)	double	Return the absolute value of x, where x is a floating point number.	double
sqrt(x)	double	Return the square root of x, where x >= 0.	double
pow(x,y)	double x, double y	Return x to the y power, $x^y$ .	double
cos(x)	double	Return the cosine of x, where x is in radians.	double
sin(x)	double	Return the sine of x, where x is in radians.	double
tan(x)	double	Return the tangent of x, where x is in radians.	double
exp(x)	double	Return the exponential of x with the base e, where e is 2.718282.	double
log(x)	double	Return the natural logarithm of x.	double
log10(x)	double	Return the base 10 logarithm of x.	double

Content Copyright Nanyang Technological University

194

The **pow()** function will take in two arguments, **x** and **y**, and returns  $x^y$ .

## MATHEMATICAL LIBRARIES - #include <math.h>

Function	Argument Type	Description	Result Type
ceil(x)	double	Return the smallest double larger than or equal to x that can be represented as an int.	double
floor(x)	double	Return the largest double smaller than or equal to x that can be represented as an int.	double
abs(x)	int	Return the absolute value of x, where x is an int.	int
fabs(x)	double	Return the absolute value of x, where x is a floating point number.	double
sqrt(x)	double	Return the square root of x, where x >= 0.	double
pow(x,y)	double x, double y	Return x to the y power, $x^y$ .	double
cos(x)	double	Return the cosine of x, where x is in radians.	double
sin(x)	double	Return the sine of x, where x is in radians.	double
tan(x)	double	Return the tangent of x, where x is in radians.	double
exp(x)	double	Return the exponential of x with the base e, where e is 2.718282.	double
log(x)	double	Return the natural logarithm of x.	double
log10(x)	double	Return the base 10 logarithm of x.	double

Content Copyright Nanyang Technological University

195

Apart from the above functions, trigonometric functions such as **sin()**, **cos()** and **tan()** that compute the sine, cosine and tangent of an angle are also provided.

## MATHEMATICAL LIBRARIES - #include <math.h>

Function	Argument Type	Description	Result Type
ceil(x)	double	Return the smallest double larger than or equal to x that can be represented as an int.	double
floor(x)	double	Return the largest double smaller than or equal to x that can be represented as an int.	double
abs(x)	int	Return the absolute value of x, where x is an int.	int
fabs(x)	double	Return the absolute value of x, where x is a floating point number.	double
sqrt(x)	double	Return the square root of x, where x >= 0.	double
pow(x,y)	double x, double y	Return x to the y power, $x^y$ .	double
cos(x)	double	Return the cosine of x, where x is in radians.	double
sin(x)	double	Return the sine of x, where x is in radians.	double
tan(x)	double	Return the tangent of x, where x is in radians.	double
exp(x)	double	Return the exponential of x with the base e, where e is 2.718282.	double
log(x)	double	Return the natural logarithm of x.	double
log10(x)	double	Return the base 10 logarithm of x.	double

Content Copyright Nanyang Technological University

196

Apart from the above functions, trigonometric functions such as **sine()**, **cos()** and **tan()** that compute the sine, cosine and tangent of an angle are also provided.

## MATHEMATICAL LIBRARIES - #include <math.h>

However, in order to use any of the mathematical functions, we need to place the preprocessor directive:

```
#include <math.h>
```

at the beginning of the program.

Content Copyright Nanyang Technological University

197

However, in order to use any of the mathematical functions, we need to place the preprocessor directive  
at the beginning of the program.

**SUMMARY**

After this lesson, you should be able to:

- Differentiate computer memory and variables
- Explain data and its types
- Define constants
- Define variables
- Explain the use of operators

Content Copyright Nanyang Technological University 198

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