CSC402
PROGRAMMING 1

LECTURE: 03 – BASIC ELEMENTS OF A COMPUTER PROGRAM

UNIVERSITI TEKNOLOGI MARA

Learning Objectives:

After completing this chapter, you will be able to:

- describe the basic components of a C++ program, including functions, special symbols, and identifiers
- use simple data types and examine the string data type
- describe how to input data into memory using input statements
- describe how to use arithmetic operators
- use of increment and decrement operators
- output results using output statements
- properly structure a program, including using comments to document a program
- use built-in function
- write a C++ program

Introduction

- Computer program: sequence of statements designed to accomplish some task
- Programming: planning/creating a program
- Syntax: rules that specify which statements (instructions) are legal
- Programming language: a set of rules, symbols, and special words
- Semantic rule: meaning of the instruction

C++ Programs

- A C++ program is a collection of one or more subprograms, called functions
- A subprogram or a function is a collection of statements that, when activated (executed), accomplishes something
- Every C++ program has one and ONLY one function called **`main`**

C++ Programs (Cont.)

- Programming instructions must be written according to syntax rules
- The smallest individual unit of a program written in any language is called a token
- Token: divided into special symbols, word symbols, and identifiers

ex: int num.

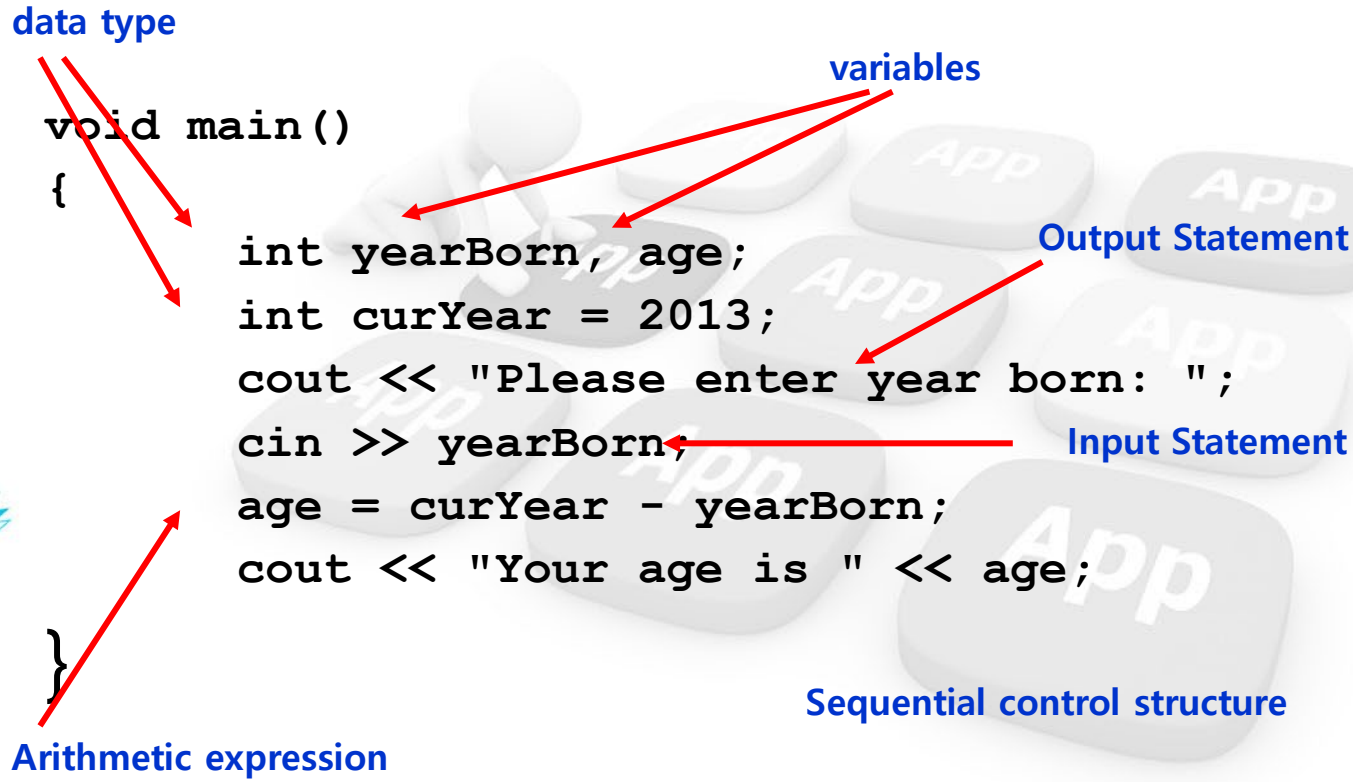
Symbols

- Special symbols
  - Include mathematical symbols and punctuation marks
  - Blank also special symbol

+	?
-	'
*	<=
/	!= not equal to
.	==
;	>=

- Word symbols
  - Another type of token
  - Reserved words or keywords
  - Reserved words: always lowercase, each considered to be single symbol with special meaning
  - Example:
    - `int`
    - `float`
    - `double`
    - `char`
    - `void`
    - `return`

A Sample Program




```
void main()
{
        int yearBorn, age;
        int curYear = 2013;
        cout << "Please enter year born: ";
        cin >> yearBorn;
        age = curYear - yearBorn;
        cout << "Your age is " << age;

}
```

A Sample Program

```
#include <iostream.h>

void main()
{
   int yearBorn;
   int curYear;
   int age;

   curYear = 2004;

   cout << "Please enter the year you were born: ";
   cin >> yearBorn;

   age = curYear - yearBorn;

   cout << "Your age is " << age;
   cout << endl;
}
```

Preprocessor directives

Main function

Variables declaration

Statements

Take note of the semicolon at the end of each variable declarations and statement (instructions)

Identifier

- Another type of token
- Identifiers: used as names for variables, constants, and functions
- Identifiers: consist of letters, digits, and underscore character (_)
- Identifiers: **must begin** with letter or underscore (best not to use underscore for portability)
- C++: case sensitive
- Some predefined identifiers (cout and cin)
- Unlike reserved words, predefined identifiers may be redefined--**but generally not good idea**

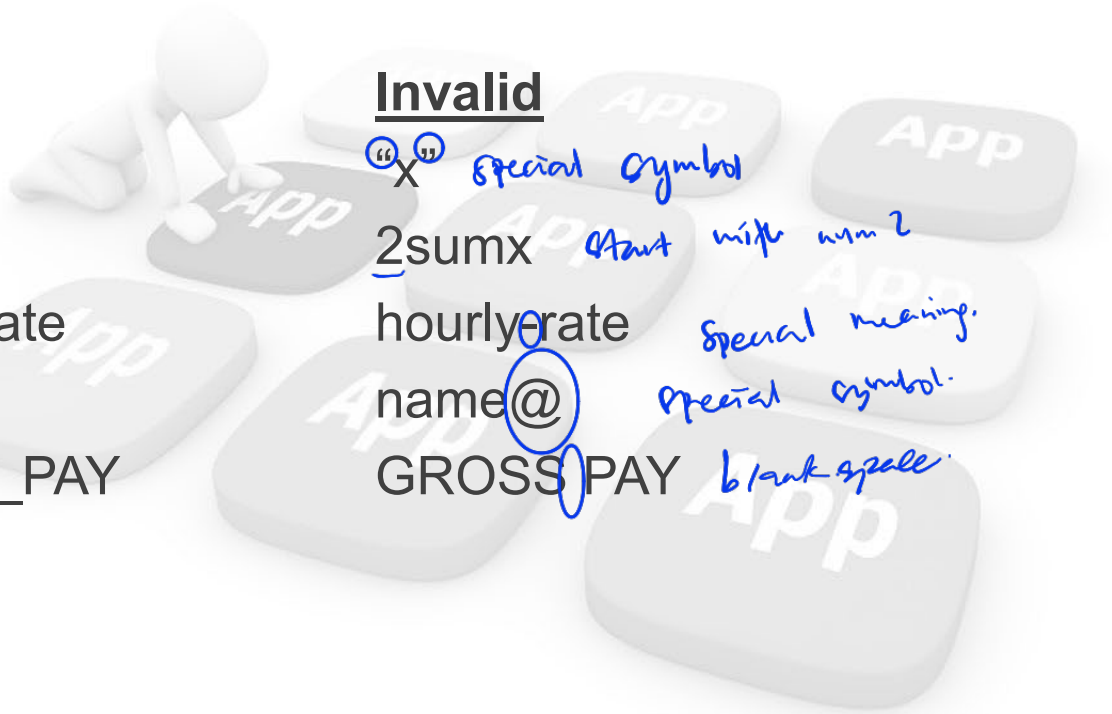
Identifier

- Although identifiers can be formed by freely combining the letters, digits and underscores, common sense tells us that we should give them suggestive names, that is, names that reflect the data items that they are going to store. Identifiers can be of any length although in practice they seldom exceed 25 characters.

Identifier

Here are some valid and invalid identifiers.

Valid	Invalid
x	"x" special symbol
sumx2	2sumx start with num 2
hourly_rate	hourly-rate special meaning.
_name	name@ special symbol.
GROSS_PAY	GROSS PAY blank space.



Identifier

C++ identifiers are case-sensitive, meaning, the lower- and uppercase letters in identifiers are treated as different characters. This means we cannot freely mix the lower- and uppercase letters to refer to the same identifier. For example, the identifiers HOURLY_RATE, Hourly_Rate, Hourly_rate and hourly_rate are all different; they refer to different memory locations.

Data Types – jenis

Data Type: set of values together with a set of operations is called a data type

C++ data can be classified into three categories:

- Simple data type
- Structured data type
- Pointers

Simple Data Types

Three categories of simple data:

- Integral: integers (numbers without a decimal)
  - int, bool, char
- Floating-point: decimal numbers
- Enumeration type: user-defined data type

float, double – titik perpuluhan

TABLE 2-2 Values and Memory Allocation for Three Simple Data Types

Data Type	Values	Storage (in bytes)
`int`	`-2147483648` to `2147483647`	4
`bool`	`true` and `false`	1
`char`	`-128` to `127`	1

int Data Type

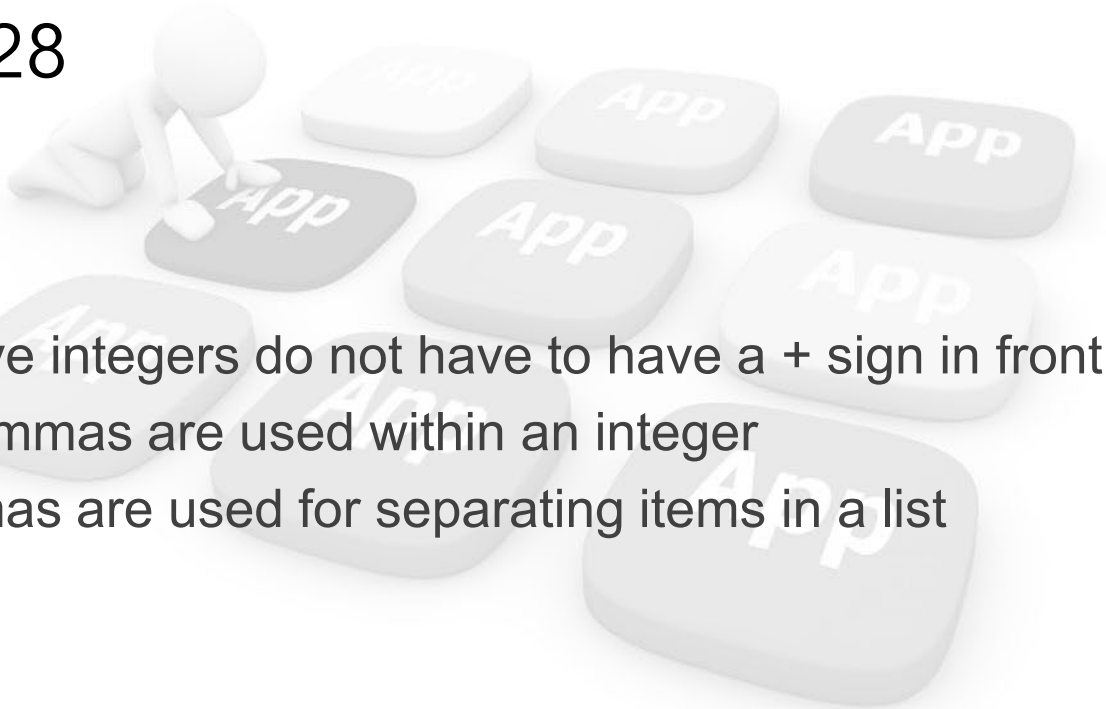
Examples:

-6728

0

78

- Positive integers do not have to have a + sign in front of them
- No commas are used within an integer
- Commas are used for separating items in a list



bool Data Type

bool type

- Has two values, **true** and **false**
- Manipulate logical (Boolean) expressions

true and **false** are called logical values

bool, **true**, and **false** are reserved words

`char` Data Type

- The smallest integral data type
- Used for characters: letters, digits, and special symbols
- Each character is enclosed in single quotes
- Some of the values belonging to char data type are: 'A', 'a', '0', '*', '+', '$', '&'
- A blank space is a character and is written ' ', with a space left between the single quotes

Floating-Point Data Types

C++ uses scientific notation to represent real numbers (floating-point notation)

TABLE 2-3 Examples of Real Numbers Printed in C++ Floating-Point Notation

Real Number	C++ Floating-Point Notation
75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-5
-1.482	-1.482000E0
7800.0	7.800000E3

Floating-Point Data Types (cont.)

- `float`: represents any real number
  - Range: -3.4E+38 to 3.4E+38
- Memory allocated for the `float` type is 4 bytes
- `double`: represents any real number
  - Range: -1.7E+308 to 1.7E+308
- Memory allocated for `double` type is 8 bytes
- On most newer compilers, data types `double` and `long double` are same

string Type

simpan lebih dari satu char

- Programmer-defined data type (not included in earlier versions of C++)
- To use string data type, string library must be included (include file)
- Sequence of zero or more characters
- Enclosed in double quotation marks (e.g., "r")
- null string: string with no characters

ex: string name; * include <cstring>

string Type (cont.)

- Each character in string has a relative position
- Position of first character is 0 (zero), 2nd character is 1, etc.
- Length of string is total number of characters (e.g., "C ++" length = 3)

Arithmetic Operators

- C++ Operators

+	addition
-	subtraction
*	multiplication
/	division
%	remainder (mod operator)

- +, -, *, and / can be used with integral and floating-point data types
- Unary operator - has only one operand
- Binary Operator - has two operands

Order of Precedence

- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- When operators are on the same level
  - Performed from left to right

Expressions

input int = output int

input float = output float.

- If all operands are integers
  - Expression is called an integral expression
- If all operands are floating-point
  - Expression is called a floating-point expression
- An integral expression yields integral result
- A floating-point expression yields a floating-point result
- Mixed expression:
  - Has operands of different data types
  - Contains integers and floating-point
  - 5.4 * 2 – 13.6 + 18 / 2

Input, Memory, and Data

Using Input:

- Data must be loaded into main memory before it can be manipulated
- Storing data in memory is two-step process:
  1. Instruct computer to allocate memory
  2. Include statements to put data into allocated memory

Input, Memory, and Data (Cont.)

→ berubah-ubah.

Allocating Memory (Constants and Variables):

- Name and define data type to store data for each memory location
- Declaration statement used to allocate memory

Input, Memory, and Data (Cont.)

- **Variables**: memory cells whose contents can be modified during program execution
- Possible to declare multiple variables in same statement (must be same type)
- **Four characteristics of variables:**
  1. Name
  2. Type
  3. Size
  4. Value

Syntax for declaring variables to allocate memory:

```
dataType identifier;
```

Examples:

```
int myVar; //declares one int variable
float myFloat, myFloat2, myFloat3; //declares three float variables
```

Ex:

```
int num1 = 20, num2 = 10
Cout << "Total" << num1+num2
<< endl;

num1 = 10, num2 = 5

Cout << "Total" << num1+num2
<< endl;
```

output:

Total = 30
Total = 15

Input, Memory, and Data (Cont.)

- Simple Data Types(Integral, Floating-point, enumeration)

Variables can hold different types of data. The various data types, the number of bits (size) required to store them are shown in Table 1.1.

Data Type	C++ keyword	Bits
integer	int	16
long Integer	long	32
short integer	short	8
unsigned integer	unsigned	16
character	char	8
floating point	float	32
double floating point	double	64
boolean	bool(true or false)	8

Input, Memory, and Data (Cont.)

- **Named Constant**: memory location whose data cannot change during program execution
- Using named constant simplifies code modification - change in declaration statement affects code globally
- In C++, const is reserved word
- *Syntax for declaring named constants to allocate memory (uses keyword const and must assign value):*

```
const dataType identifier = value;
```

- Example:

```
const double PI = 3.14;
```

Input, Memory, and Data (Cont.)

Storing Data into Variables (declaring and initializing variables):

- Variables can be declared anywhere in program
- Must be declared before can be used
- In C++, = called assignment operator
- Data stored in memory (using name of variable or constant), either using assignment statement or input statement
- C++ does not automatically initialize variables
- When variable declared, only memory is allocated
- When variable declared with no value, memory cell may contain value from setting of bits from previous use
- Expression on right side is evaluated, then its value is assigned to variable (a memory location) on left side
- Variable initialization: variable given a value at declaration **(good programming principle)**
- **Remember: assignment (=) is NOT equality (==)**

Input, Memory, and Data (Cont.)

Assignment Statement Examples:

variable = expression;

```
myVar = 10; //assigns value of 10 to myVar

int i; //allocates memory for variable i
i = i + 2; //evaluates i (right side), adds two to it, and assigns new value to memory location i (left side)
```

Input, Memory, and Data (Cont.)

Initialization Statement Examples:

int variable = expression;

```
int myVar = 10;
```
//allocates memory for variable myVar, and assigns value of 10 to myVar

//declares two double variables (myDouble and myDouble3), initializes myDouble2 with value of 25.5

```
double myDouble, myDouble2=25.5, myDouble3;
```

Input statement

- *Input (Read) Statement:*

  cin used with >> (stream extraction operator) to gather input

  When value for variable not known before program is written

  --value input through cin and >>(stream extraction operator)

- *Input (Read) Statement Examples:*
- **`cin >> variableName >> variableName. . .;`**
- **`cin >> miles;`** //computer gets value from standard input device, and places value in memory cell named miles
- **`cin >> feet >> inches;`** //input multiple values into multiple memory locations with single statement

Operators

- C++ is very rich in built-in operators. Operators trigger some computations when applied to operands in an expression.
- There are several general classes of operators. But for now, we will present just the arithmetic, relational, logical and assignment operators.

Arithmetic operators

There are seven arithmetic operators in C++ as shown below.

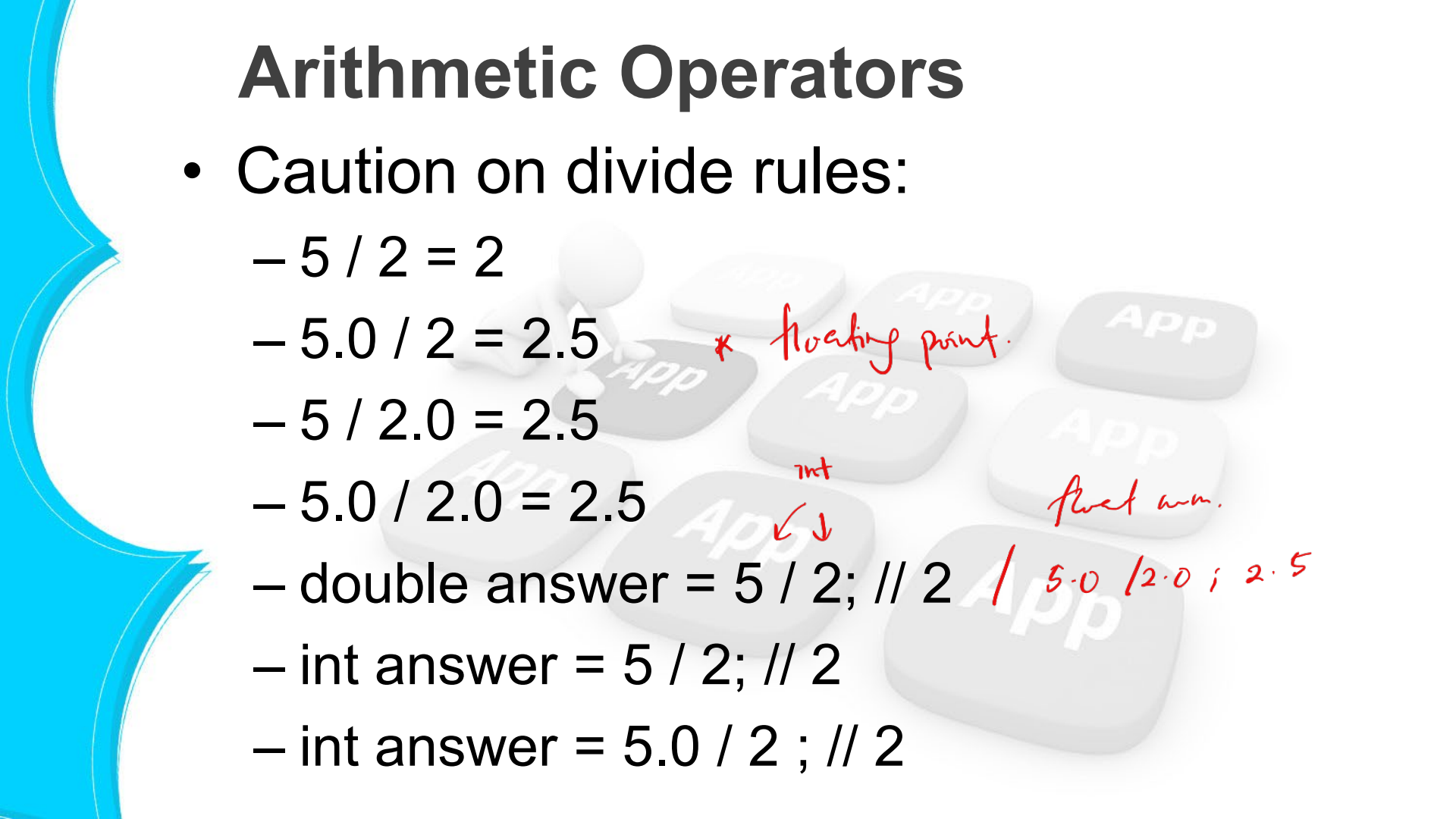
Operator	Action
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus division
++ n++;	Increment by 1 n = n+1 ex: n=10 n = 10+1 = 11
-- n--;	Decrement by 1 ex: n = 11-1 = 10

Arithmetic operators

Operator	Action	Example (no1 = 7, no2=2)	Result
+	Addition	no1 + no2	9
-	Subtraction	no1 – no2	5
*	Multiplication	no1 * no2	14
/	Division	no1 / no2	3
%	Modulus division	no1 % no2	1
++	Increment by 1	no1++ ~~no1--~~	8 ~~6~~
--	Decrement by 1	no2-- ~~no2++~~	1 ~~3~~

Arithmetic Operators

- Caution on divide rules:
  - 5 / 2 = 2
  - 5.0 / 2 = 2.5
  - 5 / 2.0 = 2.5
  - 5.0 / 2.0 = 2.5
  - double answer = 5 / 2; // 2
  - int answer = 5 / 2; // 2
  - int answer = 5.0 / 2 ; // 2

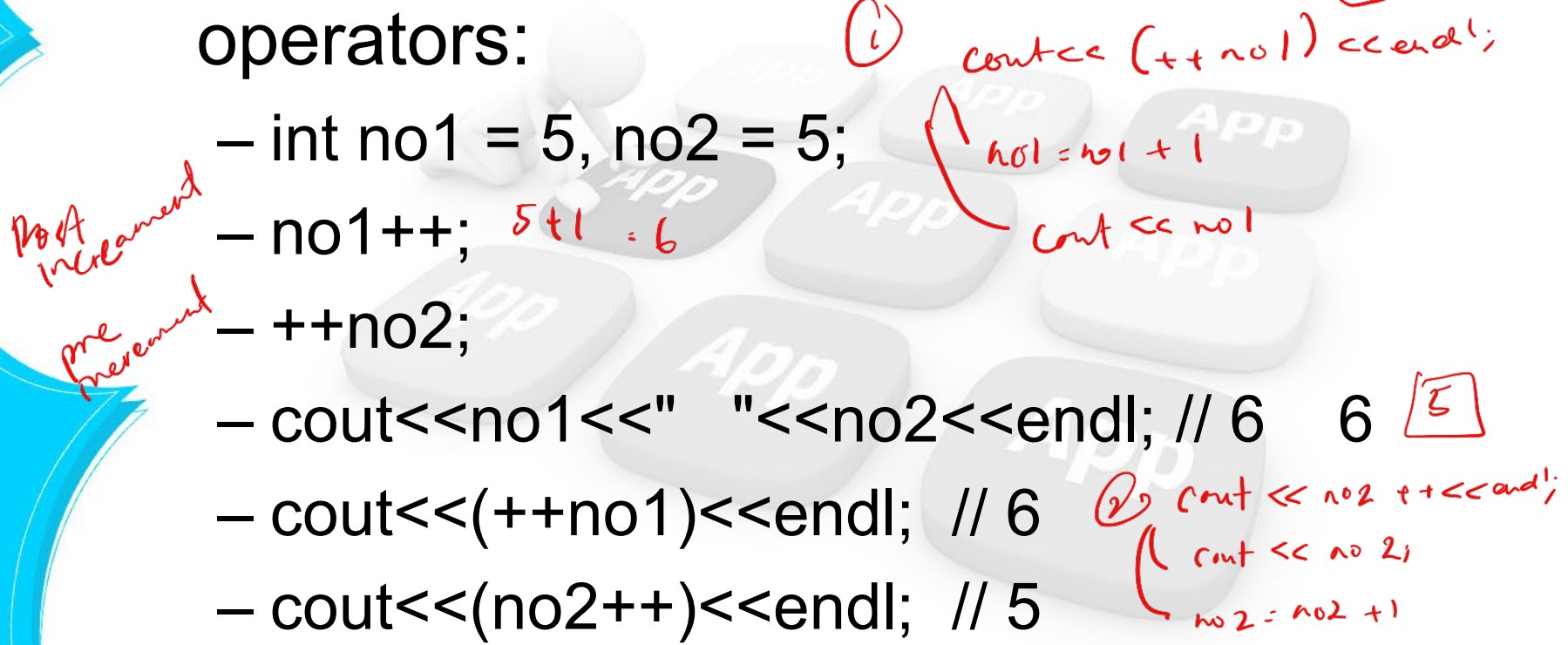


Increment & Decrement Operators

- Increment operator: increment variable by 1
- Decrement operator: decrement variable by 1
- Pre-increment: `++variable`
- Post-increment: `variable++`
- Pre-decrement: `--variable`
- Post-decrement: `variable--`

Example

- Caution on increment/decrement operators:
  - int no1 = 5, no2 = 5;
  - no1++;
  - ++no2;
  - cout<<no1<<" "<<no2<<endl; // 6 6
  - cout<<(++no1)<<endl; // 6
  - cout<<(no2++)<<endl; // 5



answer	x	y	z
?	5	3	8

answer : ++x + x * y ++ - --z;

= (5+1) + 6 * y++ - (8-1)

= 6 + 6 * y++ - 7

= 6 + 6 * 3 - 7

= 6 + 18 - 7

= 17

y = 3+1

=4

answer	x	y	z
17	6	4	7

More on Assignment Statements:

Compound assignment statements: shorthand method of assignment

Compound assignment defined: op is binary arithmetic operator, op=

Simple assignment statement variable = variable op (another variable or expression);

Rewritten as compound assignment: variable op= (another variable or expression);

Example:

+=, -=, *=, /=, and %=

x *= y; //equivalent to x = x * y;

total = total * 2;

↳ Total *= 2;

answer = answer + num;

↳ answer += num

Relational and Logical Operators

The six **relational** operators in C++ are shown next.

Operator	Meaning
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
!=	not equal

language lain:
equal < >

Relational and Logical Operators

- The results of evaluation of a relational operation is either `true` (represented by 1) or `false` (represented by 0).
- For example, if `a = 7` and `b = 5`, then `a < b` yields `0` and `a != b` yields `1`.

Relational and Logical Operators

The three **logical** operators in C++ are shown below.

combine more than one operation.

Operator	Meaning
&&	AND
||	OR
!	NOT

(age > 16 && citizen == malaysian).

ex: x < 2 , x > 10

x < 2 || x > 10

Relational and Logical Operators

The result of the logical operations on **a** and **b** are summarized in the following table.

a	b	a || b	a && b	!a
0	0	0	0	1
0	1	1	0	1
1	0	1	0	0
1	1	1	1	0

Hierarchy of operations

- There are times when you combine several operators in one statement. For example, `total = a + b / c`. The statement codes both the addition and division operators. So, the question which operation will be performed first arises; is it addition first, division later or what?
- In this situation, you need to know the hierarchy of operator precedence.
- The hierarchy of operator precedence reflects the order of evaluation that will take place.

Hierarchy of operations

The hierarchy of operator precedence from highest to lowest is summarized below.

Operator Category	Operator	Associativity
Parentheses	( )	Left to right
Unary	++ - - - +	Left to right
Arithmetic multiply, divide, remainder	* / %	Left to right
Arithmetic add and subtract	+ -	Left to right
Relational operators	< <= > >=	Left to right
Equality operators	= = !=	Left to right
Assignment operator	=	Right to left
Logical AND	&&	Left to right
Logical OR	||	Left to right

$x + y * (2+2) < x * y - 2$

$x + 4 * (2+7) < x * 4 - 2$

$6 + 4 * 9 < 6 * 4 - 2$

$6 + 36 < 24 - 2$

$42 < 22$

F → 0

x	y	z
6	4	7

ex:

```
float x = 5, answer;
answer = (x == 10);
cout << answer;
```

0

← sbb false.

Output statement

- Send output from a variable (some memory location) to the system console (the monitor) by using the function `cout` (console output) and the redirection operator <<.

- The symbol `cout` represents the output stream, a sequence of characters which are placed into the `cout` by the program when it executes and displayed on the screen.

- The redirection operator << is called the output operator or the insertion operator.

Output statement (Cont.)

For example:

1. `cout <<number1;` //insert the sequence of characters that
//represent the value of
//number1 into the stream cout
//OR, in other words, write number1

2. `cout <<"Welcome to C++ class!!!!";` //insert the string
//of characters
//into the stream cout

Escape Sequence

- An escape sequence always begins with the backslash \ and is followed by one or more special characters. One that you will use frequently is \n which means newline. For instance, the statement,
- cout<<”\nWelcome to\nC++ class!”;
- will display the following output
- Welcome to
- C++ class!

TABLE 2-4 Commonly Used Escape Sequences

	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

Built-in functions

ex: ceil(n)
ceil(2.8)
ceil(3+n)

power:
pow(a, b) = a^b

Built-in functions are also known as library functions. We need not to declare and define these functions as they are already written in the C++ libraries such as iostream, cmath etc. We can directly call them when we need.

function < programmer-defined fn / Build in fn

Commonly used header files:

<iostream> //input/output

<cmath> //math functions → complex calculation (ex: √, tan, sin, log.) — kalau x letak : error. ∴ C++ math.

<string> //string functions — dah tak relevan.

<iomanip> //formatting manipulations for input/output

Formatted Output

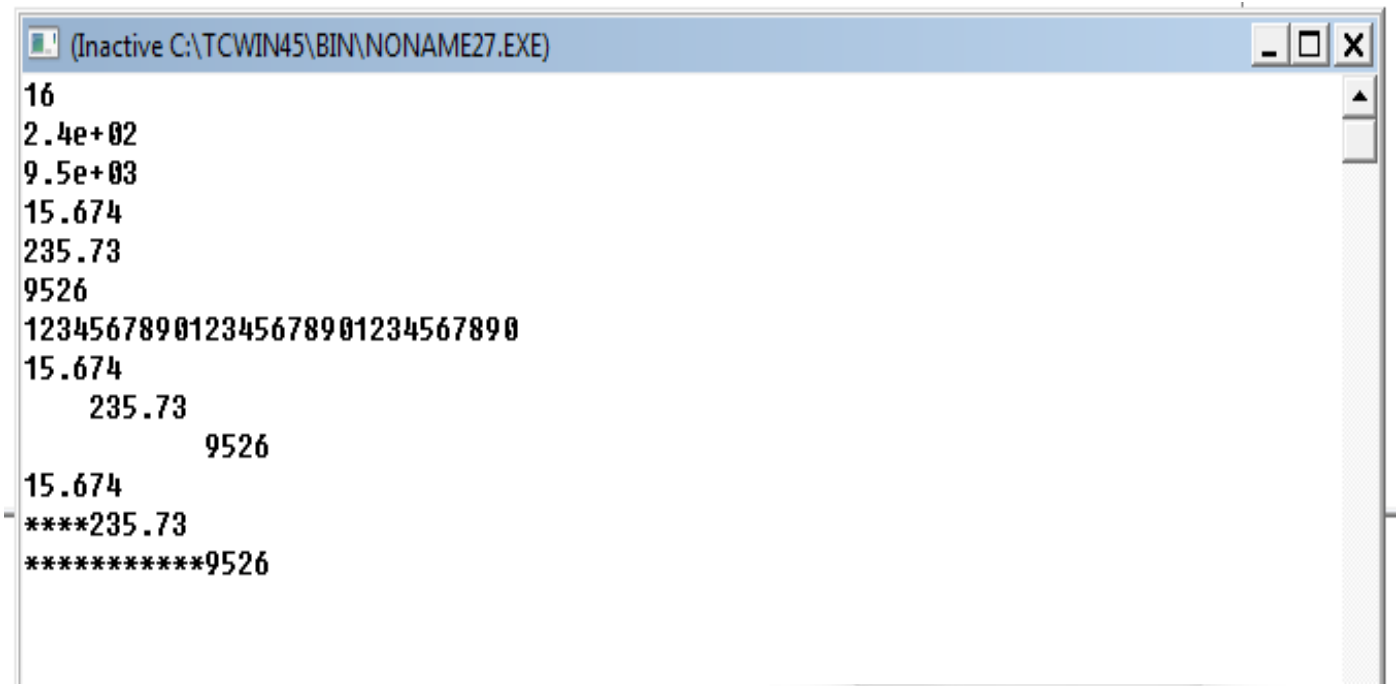
prototype : int, float.

- setprecision manipulator(to control the output of floating-point numbers)
  - `setprecision(n)`
- fixed manipulator(to output floating-point numbers in a fixed decimal format)
  - `cout<<fixed;`
- showpoint manipulator(to force the output to show the decimal point and trailing zeros)
  - `cout<<fixed<<showpoint;`
- setw manipulator(to output the value of an expression in a specific number of columns)
  - `setw(n)`
  - `cout<<setw(5)<<x<<endl;`
- setfill manipulator(to fill the unused columns with a character other than a space)
  - `cout<<setfill('*');`

Formatted Output

```
#include<iostream>
#include<iomanip>
using namespace std;
void main()
{         double x, y, z;
          x = 15.674;
          y = 235.73;
          z = 9525.9864;
          cout<< fixed << showpoint;
          cout<<setprecision(2)<<x<<"\n"<<y<<"\n"<<z<<endl;
          cout<<setprecision(5)<<x<<"\n"<<y<<"\n"<<z<<endl;
          cout<<"12345678901234567890123456789O"<<endl;
          cout<<setw(5)<< x <<endl;
          cout<<setw(10)<< y <<endl;
          cout<<setw(15)<< z <<endl;
          cout<<setfill('*')<<x<<endl;
          cout<<setw(10)<<setfill('*')<<y<<endl;
          cout<<setw(15)<<setfill('*')<<z<<endl;
}
```

Formatted Output



Formatted Output

- Be comfortable with the arithmetic operations
- + - * / %
- Remember division depends on the data type.
- cout << 3.0/4.0; //OUTPUTS 0.75
- cout << 3.0/4; //OUTPUTS 0.75
- cout << 3/4; //OUTPUTS 0
- The mod operator % gives remainder after integer
- division. Useful for detecting odd/even numbers.
- cout << 19%7; //OUTPUTS 5
- cout << 3%4; //OUTPUTS 3
- More complicated operations (sin,cos,pow,sqrt)
- require to #include <cmath> library.

Formatted Output

- *Input/Output*
- Watch the direction of your I/O push & pulls.
- Output push: `cout << x;`
- Input pull: `cin >> x;`
- Remember the basic escape characters.
- `\n` new line `\t` tab
- \\ backslash \a alert beep
- \" double quote \' single quote
- Remember the computer won't show spaces unless
- we tell it to.
- `cout << 2 << 3;` //OUTPUTS 23
- `cout << 2 << " " << 3;` //OUTPUTS 2 3

Formatted Output

- We can line items up in columns with setw(w).
- Remember it only affects the next push.
- `cout<<setw(10) << "Hello" << "Gandalf";`
- We can specify the number of decimal places with `setprecision(n)`. Affects all couts that follow.
- Remember it won't print trailing zeros unless we use the command fixed first.
- `cout << setprecision(2) << 3.12 << 3.1 << 3.0 << 3;`
- To use these functions, we need to `#include` the library `<iomanip>`

Formatted Output

- *The String Class*
- You should be comfortable with the basic member functions of the string class: length, substr, erase, insert, find.
- We concatenate (add on) pieces to the string with the plus sign +: `string s = "Hello" + s2;`
- To use any of these functions, be sure to `#include <string>`
- Remember strings are indexed from position 0 and end at position length-1.
- If we want to read in more than one word, use `getline(cin, my_string)` rather than cin>>my_string.

Formatted Output

Ch. 2: The String Class

string	Constructor. Default string is " ". Assign with string s ="word"
int s.length( )	Returns length of string s.
string s.substr (int start, int len)	Returns substring of s that starts at position start and has length len.
s.insert (int start, string s2)	Inserts string s2 into s at starting position start.
s.erase (int start, int len)	Deletes the substring from s starting at position start of length len.
int s.find (string s2, int start)	Returns starting position of substring s2 in s, starting search from position start.

Formatted Output

To *declare* a C-style string, simply declare a char array and assign it a value:

```
1  char szString[] = "string";
```

The recommended way of reading strings using cin is as follows:

```
1  char szString[255];
2  cin.getline(szString, 255);
3  cout << "You entered: " << szString << endl;
```

Formatted Output

Manipulating C-style strings

C++ provides many functions to manipulate C-style strings. For example, strcpy() allows you to make a copy of a string. However, strcpy() can cause buffer overflows! In the following program, szDest isn't big enough to hold the entire string, so buffer overflow results.

```
1 char szSource[] = "Copy this!";
2 char szDest[4];
3 strcpy(szDest, szSource); // buffer overflow!
4 cout << szDest;
```

```
1 char szSource[] = "Copy this!";
2 char szDest[50];
3 strcpy(szDest, szSource);
4 cout << szDest; // prints "Copy this!"
```

Formatted Output

It is better to use strncpy(), which takes a length parameter to prevent buffer overflow:
Other useful functions:
strcat() — Appends one string to another (dangerous)
strncat() — Appends one string to another (with buffer length check)
strcmp() — Compare two strings (returns 0 if equal)
strncmp() — Compare two strings up to a specific number of characters (returns 0 if equal)
strlen() — Returns the length of a string (excluding the null terminator)

```
  char szSource[] = "Copy this!";
1 char szDest[50];
2 strncpy(szDest, szSource, 49); // copy at most 49 character
3 s (indices 0-48)
4 szDest[49] = 0; // ensures the last character is a null ter
5 minator
  cout << szDest; // prints "Copy this!"
```

Exercises:

1. Given integer variables a=2, b=5, c=7, determine the values for each of the logical expressions given below.

a. a < 2 && y > 5

b. c >= a

c. a < b || b > c

d. a = = 25

e. c ! = 7

Exercises

2. Given integer variables x=3, y=9, z=6, determine the values for each of the following arithmetic expressions.

a. y – x + z

b. y / x * 2 * x

c. x + z / 2

d. y % z / 2

e. y + z % 4

Exercises

a. 3 / 2 + 5.5
b. 15.6 / 2 + 5
c. 4 + 5 / 2.0
d. 4 * 3 + 7 / 5 – 25.5

THANK YOU…

UNIVERSITI TEKNOLOGI MARA