# CST209

## Object-oriented Programming C++

**Prepared by Dr. Teo Bee Guan**
**Presented by Mr. Venantius Kumar Sevamalai**

# Content

1. Course Introduction
2. Lecturer contact information
3. Introduction to Programming Paradigms
4. Course Syllabus
5. Hello World Program in C++

# Course Introduction – Learning outcomes

| # | Learning outcomes |
|---|---|
| 1 | Manage the fundamentals of writing simple code with C++, such as input, output and process control |
| 2 | Perform Object-Oriented programming practice based on C++ |
| 3 | Apply knowledge to write algorithms with Object-Oriented and C++ by using class, object, inherit, and polymorphic |
| 4 | Propose solutions to computing related problems using OOP concepts via group work |

# Course Introduction – Administration

- **Teaching mode**
  - Lecture (2 hours)
  - Tutorial (2 hours)

- **Assessment and grading**
  - Continuous assessment
    i. Group Assignment (25%)
    ii. Practical Test (25 %)

  - Final assessment
    i. Final Project (50%)

# Course Introduction – Communication


- Moodle:
  - Self enrolment key: `bananarama`
- Teams Group:
  - Team code for joining Teams group: `tcj2h7p`

# Lecturer Contact Information

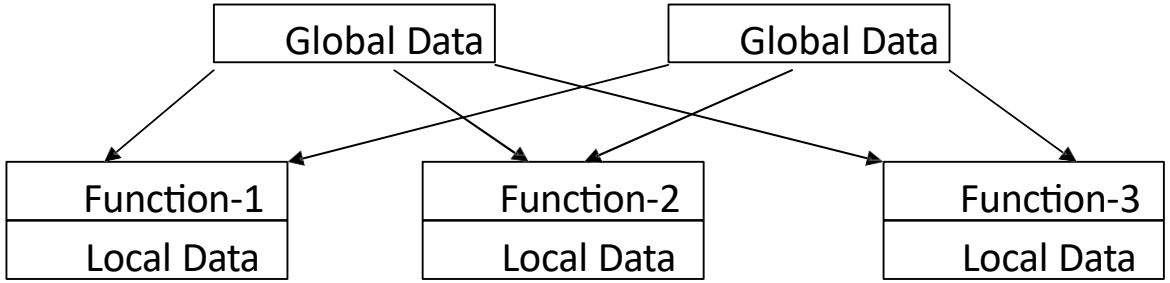
- Lecturer: Venantius Kumar Sevamalai
- Email: venantiuskumar.sevamalai@xmu.edu.my
- Consultation Hours
  - Please message in Teams Group for appointment

# Overview of Programming Paradigms

- Two main methods of writing computer programs:
  i. Procedure/Structure Programming
  ii. Object-oriented programming

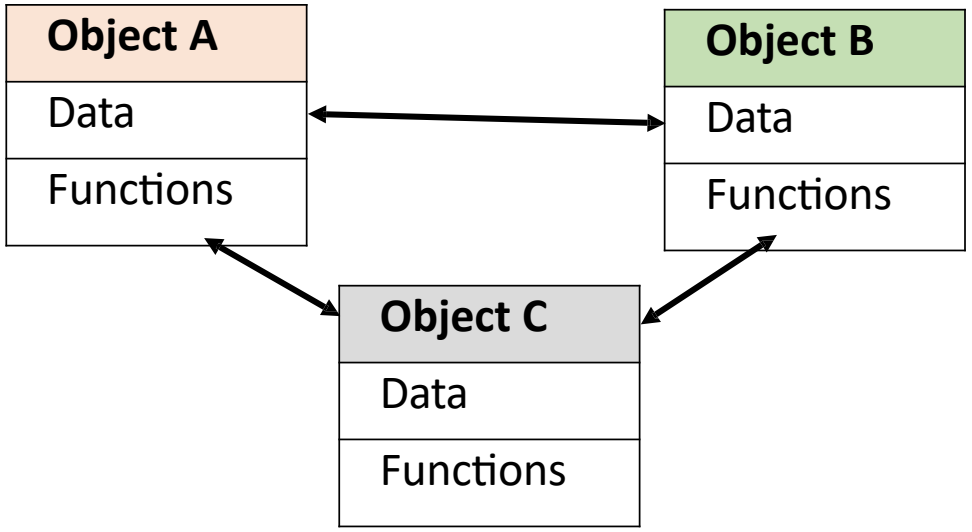
# Procedure/ Structure oriented Programming

- Conventional programming, using high level languages such as COBOL, FORTRAN, and C, is commonly known as procedure-oriented programming (POP).
- In the procedure-oriented approach, the problem is viewed as a sequence of things to be done, such as reading, calculating, and printing. Several functions are written to accomplish these tasks.
- The primary focus is on functions.



# Object Oriented Programming

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.



# Topics of Object Oriented Programming (OOP)

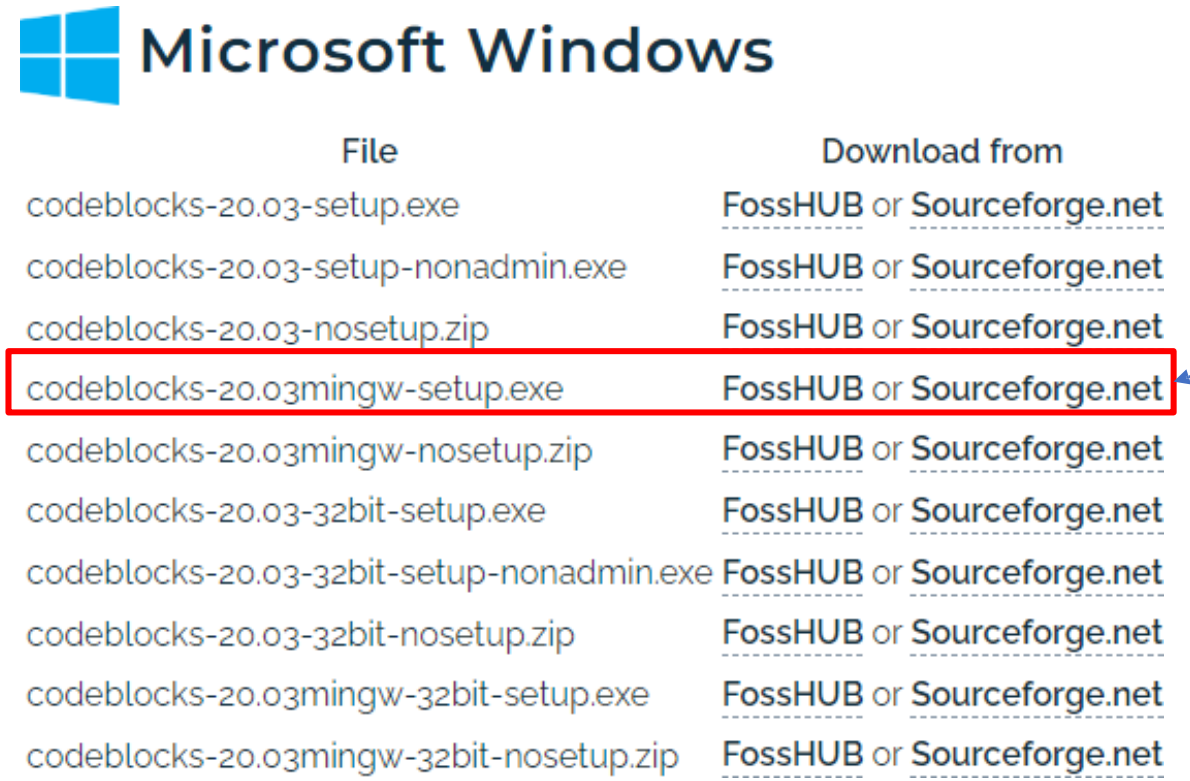
We will cover the topics below throughout the course:

- Classes
- Objects
- Encapsulation
- Inheritance
- Polymorphism

# Course Introduction – Syllabus

| Section | Title |
|---|---|
| Part 1 | C++ Fundamental (c ++ programming environment, c ++ class library, c ++ extension) |
| Part 2 | Concept of Object-Oriented Programming<br>• Encapsulation<br>• Class<br>• Object |
| Part 3 | Inheritance |
| Part 4 | Polymorphism |

CodeBlock - https://www.codeblocks.org/downloads/binaries/




Choose this installer

# A Simple C++ Program

```
#include <iostream> //include header file
using namespace std;
int main()
{
      cout << "Hello World"; // C++ statement
      return 0;
}
```

- **iostream** is just like we include **stdio.h** in c program.
- It contains declarations for the identifier **cout** and the insertion operator **<<**.
- **iostream** should be included at the beginning of all programs that use input/output statements.

**Practice: Code Example 1**

# A Simple C++ Program

```
#include <iostream> //include header file
using namespace std;
int main()
{
      cout << "Hello World"; // C++ statement
      return 0;
}
```

- A namespace is a declarative region.
- A **namespace** is a part of the program in which certain names are recognized; outside of the namespace they're unknown.
- namespace defines a scope for the identifies that are used in a program.
- **using** and **namespace** are the keywords of C++.

# A Simple C++ Program

```
#include <iostream> //include header file
using namespace std;
int main()
{
      cout << "Hello World"; // C++ statement
      return 0;
}
```

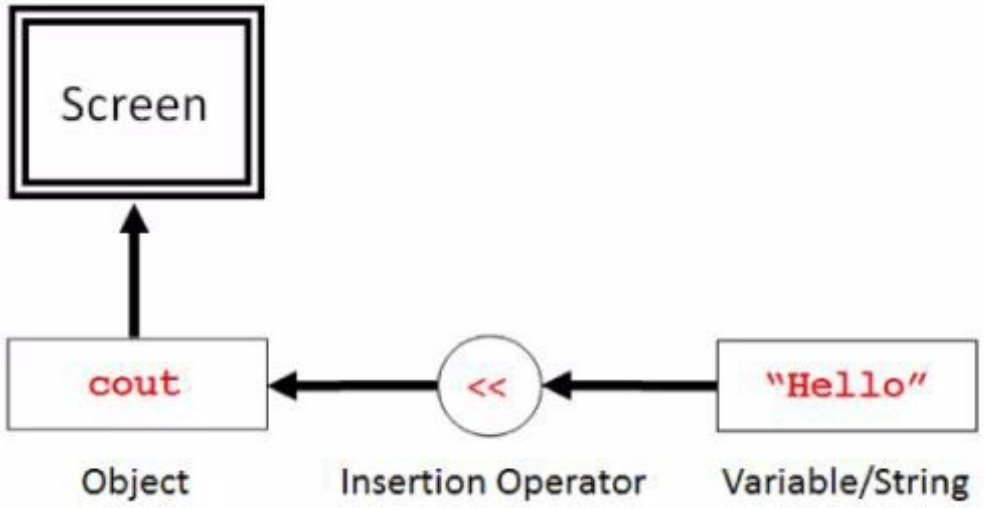
- **std** is the namespace where ANSI C++ standard class libraries are defined.
- Various program components such as **cout**, **cin**, **endl** are defined within **std** namespace.
- If we don't use the **using** directive at top, we have to add the **std** followed by **::** in the program before identifier.

```
std::cout << "Hello World";
```

# A Simple C++ Program

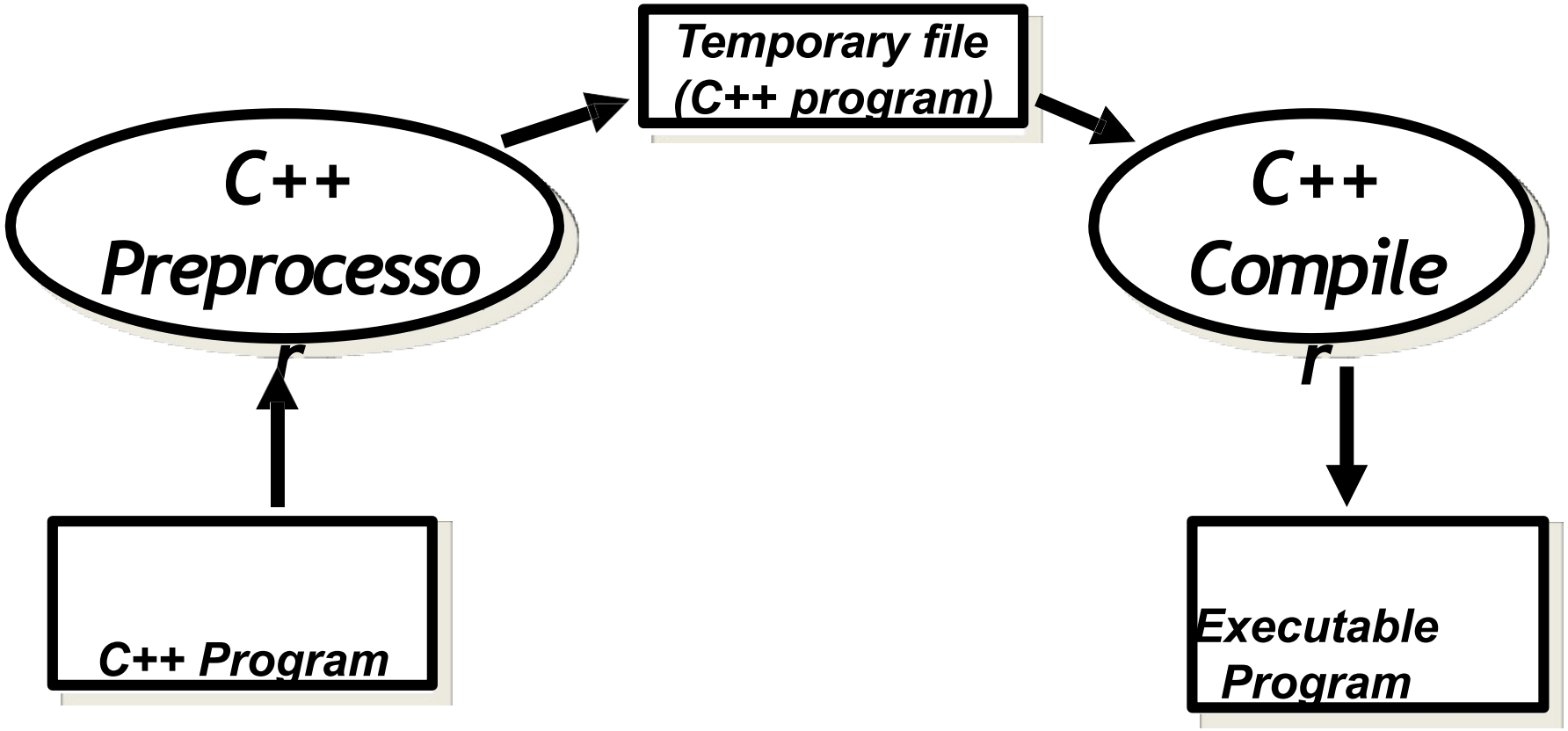
```
cout << "Hello World";
```

- The operator << is called the insertion operator.
- It inserts the contents of the variable on its right to the object on its left.
- The identifier **cout** is a predefined object that represents standard output stream in C++.
- Here, Screen represents the output. We can also redirect the output to other output devices.



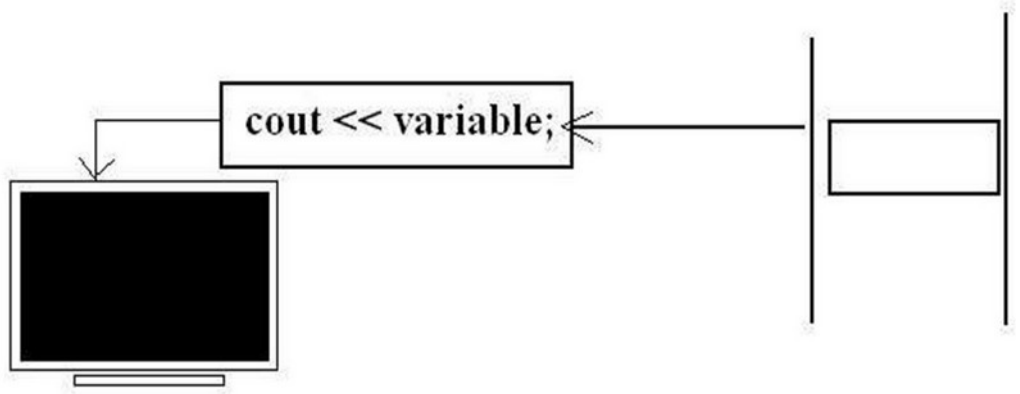

Output Using Insertion Operator

# Mechanism of C++ Preprocessing



# C++ Input and Output

## Output Statement




```
cout << "Welcome to c++";
```


**Practice: Code Example 2, 3, 4**

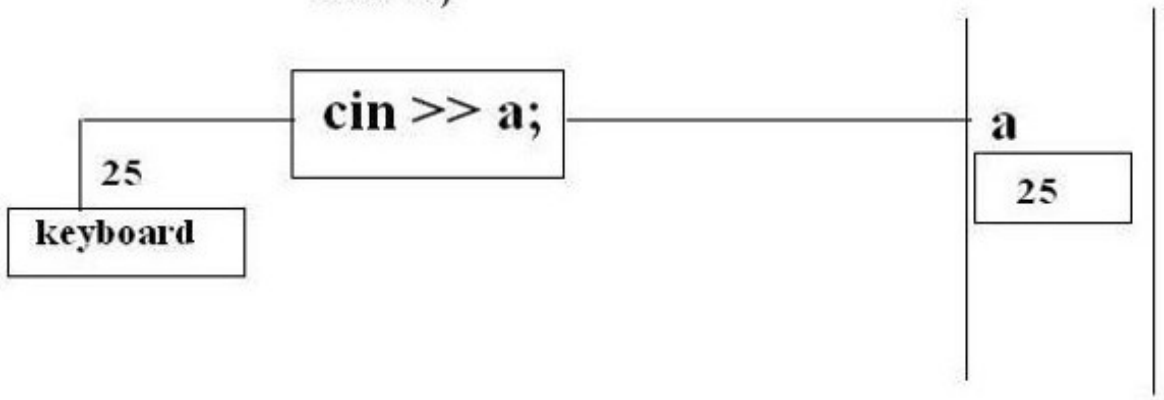
# C++ Input and Output

## Input Statement

```
cin >> variable ;
```

example:

```
int a;
cin >> a;
```




## Output Statement

```
a = 100;
cout << a;
```


```
a = 10;
cout << "Value of a is " << a;
```


**Practice: Code Example 5**

# Reserved Keywords

- Has predefined functionality
- Written only in lowercase
- Total number of keywords are updated from time to time

| | | | |
|---|---|---|---|
| asm | dynamic_cast | new | template |
| auto | else | operator | this |
| bool | enum | private | throw |
| break | extern | protected | true |
| case | false | public | try |
| catch | float | register | typedef |
| char | for | reinterpret_cast | typeid |
| class | friend | return | union |
| const | goto | short | unsigned |
| const_cast | if | signed | using |
| continue | inline | sizeof | virtual |
| default | int | static | void |
| delete | long | static_cast | volatile |
| do | mutable | struct | wchar_t |
| double | namespace | switch | while |

# Identifiers

- Programmer-designed tokens
- Meaningful & short
- Long enough to understand
- C++ rules for Identifiers
  - alphabets, digits, underscore
  - should not start with digits.
  - Case sensitive
  - Unlimited length
  - Declared anywhere

| IDENTIFIER | VALID? | REASON IF INVALID |
|---|---|---|
| `totalSales` | Yes | |
| `total_Sales` | Yes | |
| `total.Sales` | No | Cannot contain `.` |
| `4thQtrSales` | No | Cannot begin with digit |
| `totalSale$` | No | Cannot contain `$` |

# Literals

- Sequence of char that represents constant values to be stored in variables
- C++literals are:
  - Integer literals: 1,2,456,0xffff
  - Floating_point literals: 4.67,3.14E-05
  - Character literals: ‘A’, ‘B’
  - String literals: “ABC”, “TOTAL”

**Practice: Code Example 6**

# Literals (Symbolic Constants)

- Using **const** qualifier  ex: const int size=10;
- Using **#define** pre-processor ex: #defines X=0
- Using **enum** keyword  ex: enum{X,Y,Z};

**Practice: Code Example 7**

# Operators

- Is a symbol that takes more than one operand and operates on them to produce a result.
  - Arithmetic
  - Relational
  - Logical
  - Assignment
  - increment/Decrement
  - conditional
  - scope resolution(::)
  - special operators: new, delete, endl, setw

**Practice: Code Example 8**

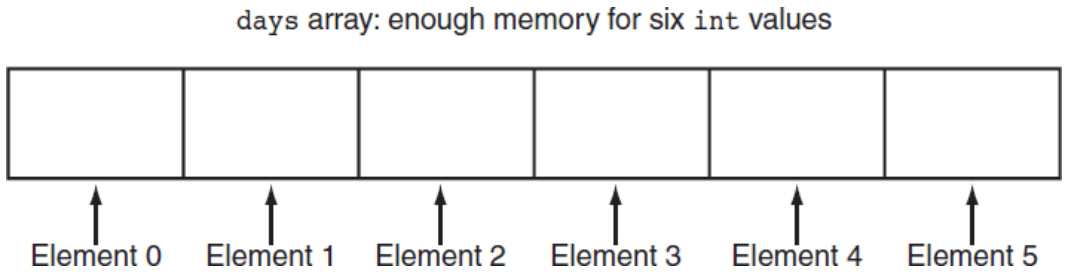
# Separators

- Symbols used to indicate where groups of code are divided & arranged
- C++ separators:
- **()parentheses** .. Methods, precedence in exp
- **{} braces ..** Arrays init., block of codes, scopes
- **; semicolon**
- **,comma..** Separate multiple identifiers, chain  more than one stmt
- **. Period..** Data members, methods
- **[]Brackets..** Array referencing/dereferencing

**Practice: Code Example 9**

# Array

- An array allows you to store and work with multiple values of the same data type.
- The values are stored together in consecutive memory locations.




```
float temperatures[100];     // Array of 100 floats
string names[10];            // Array of 10 string objects
long units[50];              // Array of 50 long integers
double sizes[1200];          // Array of 1200 doubles
```

Syntax to declare an array

**Practice: Code Example 10, 11**

See you next class