Programming 1 (C++)

Introduction to C++

Profile Saya

Wirawan Andi Nugroho, S.T., M.Kom

S1 Teknik Mesin, ITS Surabaya
S2 Manajemen Sistem Informasi, BINUS Jakarta

Lean Manager, PT. Pepperl+Fuchs Bintan (2016-sekarang)
Lecturer, STTI Tanjungpinang (2023-sekarang)

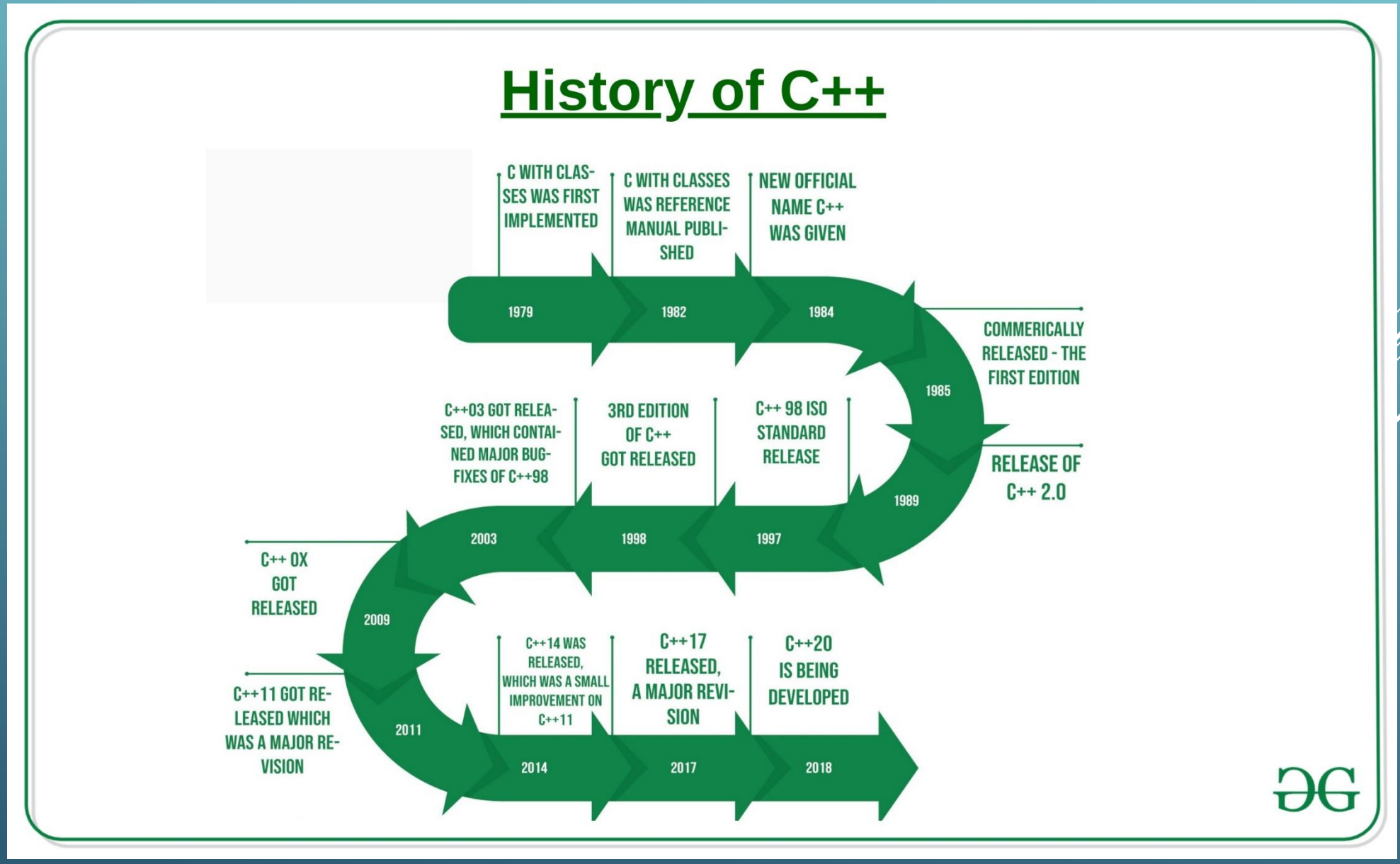
Wirawan.andi.nugroho2@gmail.com

081267516457

wirawan-andi-nugroho-lean-sixsigma/

OVERVIEW OF C++

Definition: C++ is a general-purpose programming language developed as an **extension of the C programming language** with additional features such as **classes** and **object-oriented programming**.



IMPORTANCE AND APPLICATION OF C++

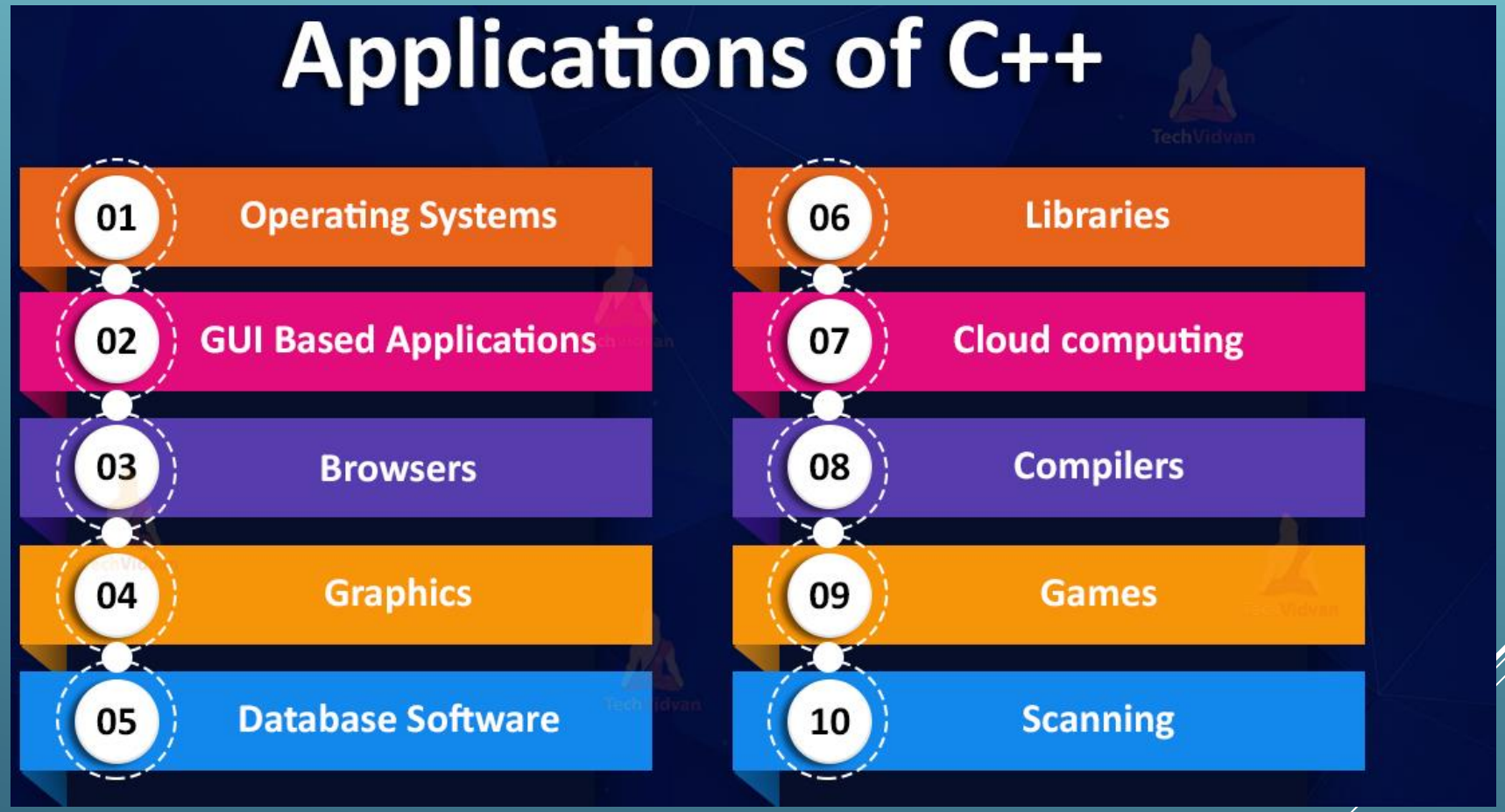
Windows
macOS
Linux

Adobe i.e.
Photoshop,
Illustrator,
Adobe Premier

Google Chrome,
Mozilla

3D animation,
modeling,
image processing,
real-time simulations

MySQL,
Postgres,
Redis, and Oracle



Dev-C++,
Apple C++,
Clang C++,
MINGW

KEY FEATURES OF C++

1. **Object-Oriented Programming (OOP):**

Definition: C++ supports the principles of OOP, such as encapsulation, inheritance, and polymorphism.

Importance: OOP enables better organization and abstraction of code, making it more modular and easier to maintain.

2. **Efficiency:**

Low-Level Features: C++ provides features like pointers and manual memory management, allowing developers to control system resources efficiently.

Inline Assembly: C++ allows the use of inline assembly, facilitating optimization for specific hardware architectures.

3. **Versatility:**

Multi-Paradigm: C++ is a multi-paradigm programming language, supporting procedural, object-oriented, and generic programming styles.

Standard Template Library (STL): The STL provides a collection of generic classes and functions, enhancing code reusability and efficiency.

KEY FEATURES OF C++

4. **Portability:**

Platform Independence: C++ programs can be compiled on different platforms without modification, promoting code portability.

Compatibility with C: C++ is backward compatible with C, allowing integration of existing C code.

5. **Standardization:**

ISO/IEC Standard: C++ is an internationally standardized language, ensuring consistency and compatibility across different compilers and platforms.

6. **Community Support:**

Active Community: C++ has a large and active community of developers, providing a wealth of resources, libraries, and frameworks.

CLASSES

1. Classes : A class is a blueprint for creating objects. It defines a template for objects, specifying their attributes (data members) and behaviors (member functions).

```
// Declaration of the class
class Car {
public:
    // Data members (attributes)
    std::string brand;
    int year;
    double price;

    // Member functions (behaviors)
    void displayInfo();
};

// Definition of the member function outside the class
void Car::displayInfo() {
    std::cout << "Brand: " << brand << "\nYear: " << year << "\nPrice:
}
```

- In this example, we define a class **Car** with data members (**brand**, **year**, **price**) and a member function (**displayInfo**). The function **displayInfo** is defined outside the class.

CLASSES

2. Creating Object

```
// Creating objects of the Car class
Car car1, car2;

// Accessing and modifying object attributes
car1.brand = "Toyota";
car1.year = 2022;
car1.price = 25000.0;

car2.brand = "Honda";
car2.year = 2021;
car2.price = 22000.0;
```

3. Member Function and Encapsulation : Member functions define the behaviors of objects and can be called on instances of the class.

```
// Calling member functions
car1.displayInfo();
car2.displayInfo();
```

OBJECT-ORIENTED PROGRAMMING

1. Encapsulation:

Encapsulation bundles the data (attributes) and the functions (behaviors) that operate on the data within a single unit, i.e., a class.

Example: The Car class encapsulates the attributes (brand, year, price) and the displayInfo function.

2. Inheritance:

Inheritance allows a class (subclass/derived class) to inherit properties and behaviors from another class (base class).

```
// Base class
class Vehicle {
public:
    int wheels;
};

// Derived class inheriting from Vehicle
class Car : public Vehicle {
public:
    // Additional attributes and functions specific to Car
    std::string brand;
    void startEngine();
};
```

OBJECT-ORIENTED PROGRAMMING

3. **Polymorphism:**

Polymorphism allows objects of different types to be treated as objects of a common type.

```
// Polymorphic behavior through function overloading
void accelerate(Vehicle& vehicle) {
    std::cout << "Accelerating with " << vehicle.wheels << " wheels.\n";
}

// Usage
Car myCar;
myCar.wheels = 4;
accelerate(myCar);  // Polymorphic call
```

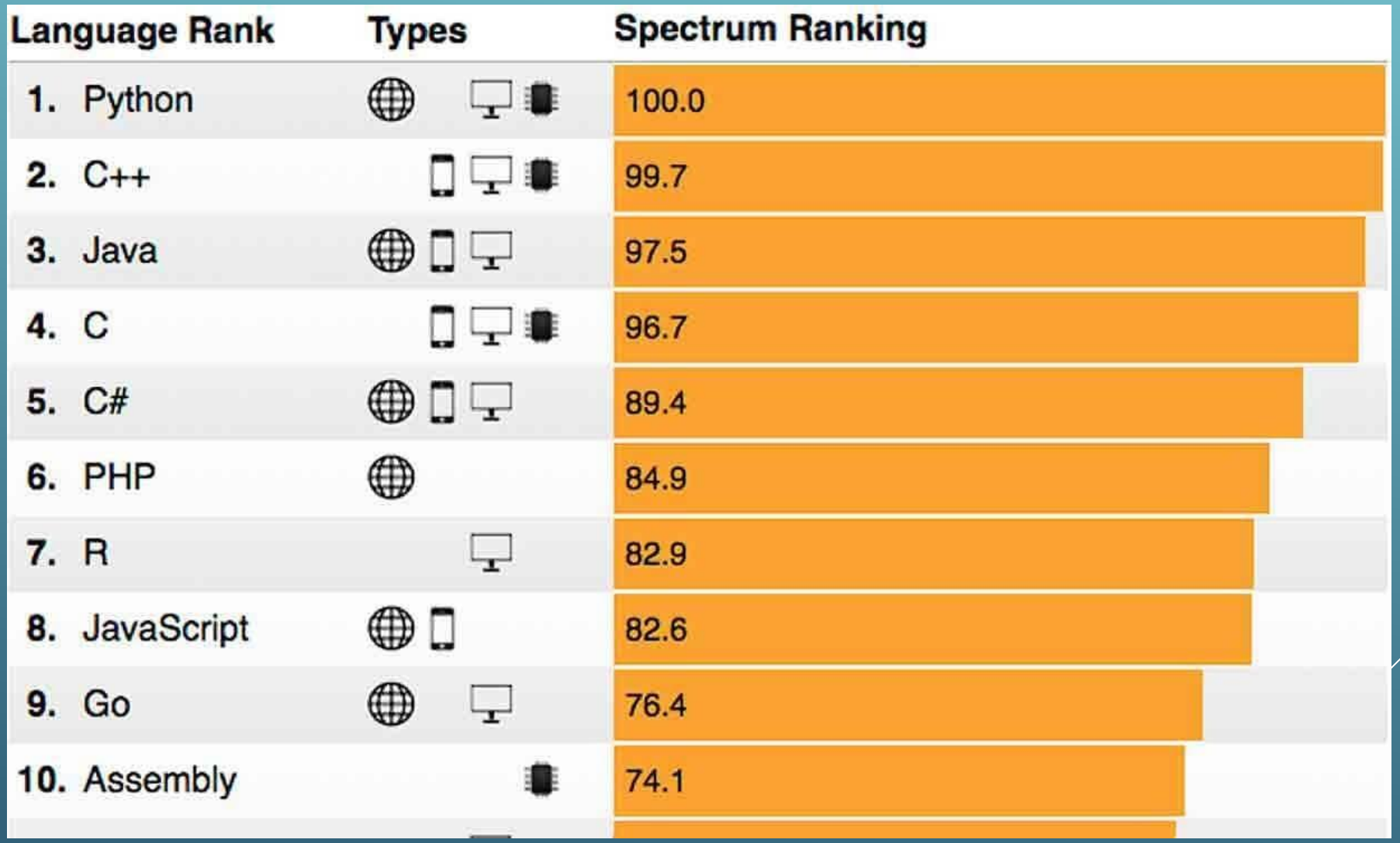
4. **Abstraction:**

Abstraction involves hiding the complex implementation details and showing only the essential features of an object.

```
// Abstract class
class Shape {
public:
    virtual void draw() = 0;  // Pure virtual function
};

// Concrete class implementing Shape
class Circle : public Shape {
public:
    void draw() override {
        std::cout << "Drawing a circle.\n";
    }
};
```

COMPARISON WITH C AND OTHER LANGUAGE



COMPARISON WITH C AND OTHER LANGUAGE

1. C vs. C++:

OOP Support: C++ introduces OOP concepts (classes, objects) that are absent in C.

Function Overloading: C++ allows function overloading, enabling multiple functions with the same name but different parameters.

Standard Template Library (STL): C++ includes the STL, providing generic algorithms and containers.

Namespace: C++ introduces namespaces to avoid naming conflicts, which are not present in C.

2. C++ vs. Other Programming Languages:

Java: C++ is a lower-level language with manual memory management, while Java is more abstract with automatic memory management (garbage collection).

Python: C++ is compiled and statically typed, whereas Python is interpreted and dynamically typed.

C#: C# shares similarities with C++ but is primarily used in the Microsoft ecosystem. C++ is more versatile and platform-independent.

COMPARISON WITH C AND OTHER LANGUAGE

3. Performance Comparison:

Efficiency: C++ is often considered more efficient than higher-level languages due to its control over system resources and low-level features.

Execution Speed: C++ programs generally have faster execution speeds compared to interpreted languages like Python.

4. Language Popularity and Usage:

Industry Use: C++ is commonly used in system programming, game development, embedded systems, and performance-critical applications.

Learning Curve: C++ might have a steeper learning curve compared to some higher-level languages.

COMPILER

https://replit.com

https://www.onlinegdb.com/online_c++_compiler

https://godbolt.org/

BASIC SYNTAX RULES

Case Sensitivity:

C++ is case-sensitive. This means that Variable and variable are considered different identifiers.
Comments:

Comments are used to add explanations or notes to the code and are not executed by the compiler.
Single-line comments: // This is a single-line comment.
Multi-line comments:

```
/*
   This is a multi-line comment.
   It can span multiple lines.
*/
```

BASIC SYNTAX RULES

Semicolons:

Statements in C++ must end with a semicolon (;).
Example: int x = 5;
Braces (Curly Brackets):

Used to define blocks of code.
Opening brace { must be followed by a closing brace }.
Example:

```
int main() {
    // code here
    return 0;
}
```

BASIC SYNTAX RULES

Indentation:

Although not required by the compiler, proper indentation enhances code readability.
It is common practice to indent code within blocks.
Variables and Data Types:

Variables must be declared before they are used.
Each variable must have a specific data type.
Example:

```
int age;  // Declaration
age = 25; // Assignment
```

BASIC SYNTAX RULES

Keywords:

Certain words are reserved for specific purposes and cannot be used as identifiers.
Examples: int, double, if, for, while, return, etc.
Functions:

Functions are declared with a return type, a name, and parentheses.
Parentheses may include parameters.
Example:

```
int add(int a, int b) {
    return a + b;
}
```

BASIC SYNTAX RULES

Operators:

C++ includes various operators such as arithmetic (+, -, *, /), relational (<, >, ==), logical (&&, ||), etc.
Example:

```
int sum = 5 + 3;  // Arithmetic operator
bool isEqual = (sum == 8);  // Relational operator
```

BASIC SYNTAX RULES

Conditional Statements:

if, else if, and else are used for conditional execution.
Example:

```
if (x > 0) {
    // code when x is positive
} else if (x == 0) {
    // code when x is zero
} else {
    // code when x is negative
}
```

BASIC SYNTAX RULES

Loops:

for, while, and do-while are used for looping.
Example:

```
for (int i = 0; i < 5; ++i) {
    // code to be repeated
}
```

DATA TYPES

1. Built-in Data Types:
    a. Primitive Data Types:
        Integer Types:
        - int: Represents signed integers (whole numbers) with a typical size of 4 bytes.
        - short: Represents shorter signed integers.
        - long: Represents longer signed integers.
        - long long: Represents very long signed integers.

        Floating-point Types:
        - float: Represents single-precision floating-point numbers.
        - double: Represents double-precision floating-point numbers (more precise than float).
        - long double: Represents extended-precision floating-point numbers.

        Character Type:
        - char: Represents a single character. It is typically 1 byte in size.
        - Boolean Type: bool: Represents Boolean values (true or false).

DATA TYPES

```
// Integer Types
int integerNumber = 42;
short shortNumber = 32767;
long longNumber = 2147483647L;
long long veryLongNumber = 9223372036854775807LL;

// Floating-point Types
float floatNumber = 3.14f;
double doubleNumber = 3.1415926535897793;
long double extendedDouble = 3.14159265358979323L;

// Character Type
char character = 'A';

// Boolean Type
bool isTrue = true;
bool isFalse = false;
```

DATA TYPES

b. Derived Data Types:

Pointers:

- int*: Pointer to an integer.
- double*: Pointer to a double.
- char*: Pointer to a character.

References:

- int&: Reference to an integer.
- double&: Reference to a double.
- char&: Reference to a character.

```
// Pointers
int* pointerToInt = nullptr;
double* pointerToDouble = nullptr;
char* pointerToChar = nullptr;

// References
int originalInt = 10;
int& referenceToInt = originalInt;

// Arrays
int numbers[5] = {1, 2, 3, 4, 5};
```

DATA TYPES

3. Modifiers:

a. Modifiers for Integer Types:

- signed: Represents both positive and negative values (default for int).
- unsigned: Represents only non-negative values (no sign bit).

b. Modifiers for Floating-point Types:

- float: Single-precision floating-point.
- long double: Extended-precision floating-point.

c. Size Modifiers:

- short, long, long long: Used with integer types to specify the size.

```
// Signed and Unsigned Modifiers
signed int signedNumber = -10;
unsigned int unsignedNumber = 20;

// Size Modifiers
short shortInteger = 32767;
long longInteger = 2147483647L;
long long veryLongInteger = 9223372036854775807LL;
```

DATA TYPES

4. Typedef:

Used to create an alias for an existing data type.

Example:

```
typedef double Distance;
Distance length = 10.5;
```

5. Auto and decltype:

auto: Automatically deduces the data type of a variable during compilation.

decltype: Returns the data type of an expression.

Example:

```
auto x = 5;             // x is deduced to be int
decltype(x) y = x;      // y has the same type as x
```

6. Dynamic Memory Allocation:

new and delete operators are used for dynamic memory allocation.

Example:

```
int* dynamicArray = new int[10];
delete[] dynamicArray;
```

THANK YOU