C++ Programming

C++

General Purpose Programming Language

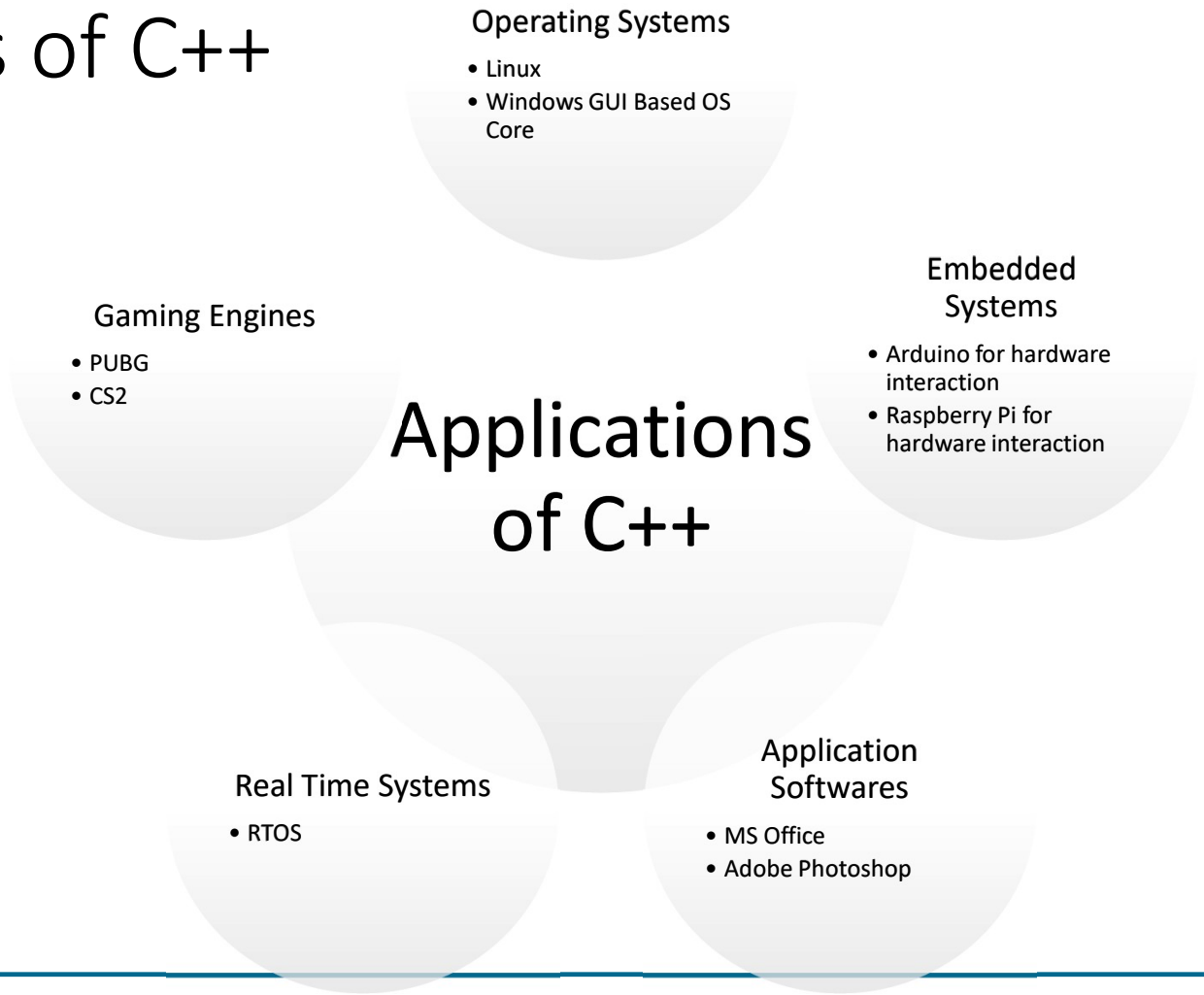
Superset of C Programming

Developed by Bjarne Stroustrup at AT&T Bell Laboratory in 1979

Developed for fast speed, low level memory management

High Level Programming Language

Applications of C++



Features of C++

Simple

- Programs can be broken down into logical parts
- Rich library support
- Variety of data types

Machine Independent

- Can run on any machine
- Requires only a suitable compiler

Low-level Access

- Access to system resources
- Ideal for system-level programming
- Enables writing highly efficient code

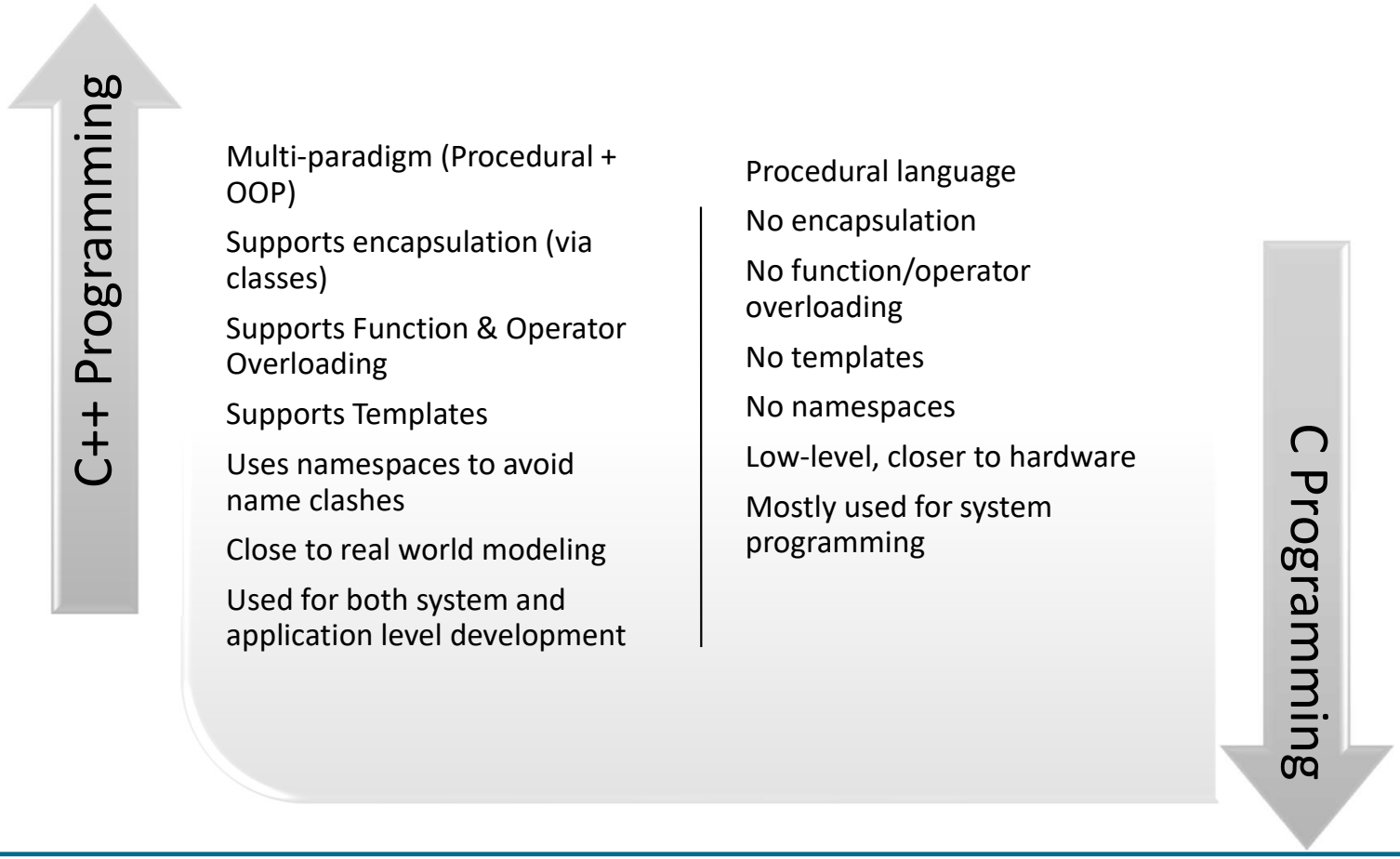
Fast Execution Speed

- Minimal processing overhead
- One of the fastest high-level languages
- Suitable for performance-critical applications

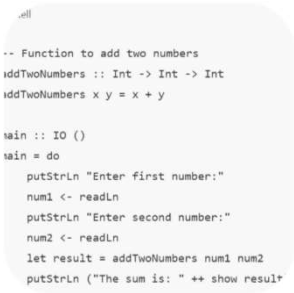
Object-Oriented

- Supports OOP features like classes, inheritance, polymorphism
- Makes code **maintainable** and **extensible**
- Well-suited for building large-scale applications

C++ Vs C Programming



TYPES OF PROGRAMMING LANGUAGES

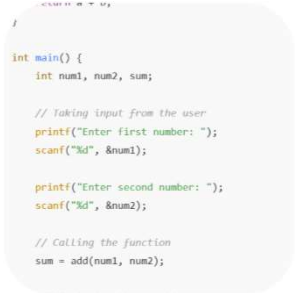


```
-- Function to add two numbers
addTwoNumbers :: Int -> Int -> Int
addTwoNumbers x y = x + y

main :: IO ()
main = do
    putStrLn "Enter first number:"
    num1 <- readLn
    putStrLn "Enter second number:"
    num2 <- readLn
    let result = addTwoNumbers num1 num2
    putStrLn ("The sum is: " ++ show result
```

Functional Programming

- Based on mathematical functions to perform computation
- Eg: Scala, Haskell, and F#.



```
int main() {
    int num1, num2, sum;

    // Taking input from the user
    printf("Enter first number: ");
    scanf("%d", &num1);

    printf("Enter second number: ");
    scanf("%d", &num2);

    // Calling the function
    sum = add(num1, num2);
```

Procedural Programming

- Uses a series of functions / procedures to perform computation
- Eg: C, Pascal, Fortran, Ada, and Basic

```
// Function to take input
void getNumbers() {
    cout << "Enter first number: ";
    cin >> num1;
    cout << "Enter second number: ";
    cin >> num2;
}

// Function to add two numbers
int add() {
    return num1 + num2;
}
};

int main() {
    Adder obj; // Creating an object of the class

    obj.getNumbers(); // Taking input
    int sum = obj.add(); // Calling function to add numbers

    // Displaying the result
```

Object-Oriented programming

- Every component of program is considered as object and this is used to perform desired action
- Eg: C++ , Java

```
    self.num1 = 0
    self.num2 = 0

def get_numbers(self):
    self.num1 = int(input("Enter first number: "))
    self.num2 = int(input("Enter second number: "))

def add(self):
    return self.num1 + self.num2


# Creating an object of the class
obj = Adder()

# Taking input
obj.get_numbers()

# Calculating sum
_result = obj.add()
```

Scripting Language

- Designed for communicating with other programming languages
- Eg: Python, JavaScript

C++ Versions

C++98
- First standardized version
- Introduced STL (Standard Template Library)
- Basis for future versions

C++03
- Bug fixes and small improvements over C++98
- Technical refinement

C++11 ("Modern C++")
- Auto keyword, nullptr
- Range-based for loops
- Lambda expressions
- Smart pointers

C++14
- Minor enhancements over C++11
- Binary literals, generic lambdas
- std::make_unique

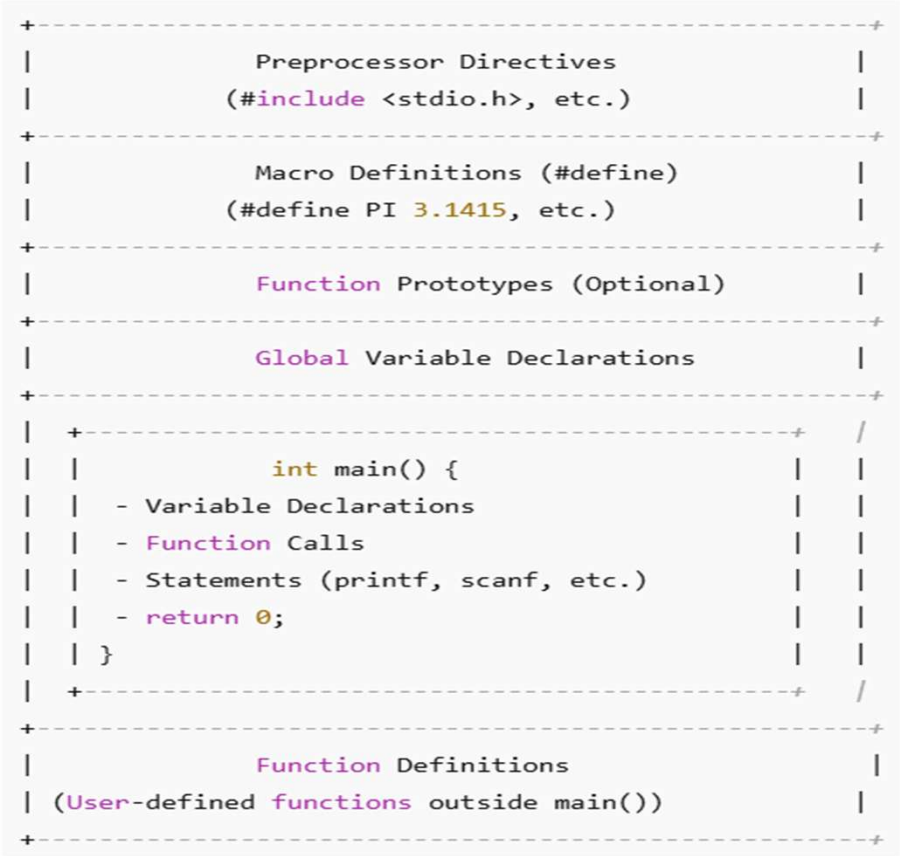
C++17
- Structured bindings
- if constexpr, std::optional
- Parallel STL algorithms

C++20
- Concepts, Coroutines
- Ranges library, consteval
- Modules (partial support)

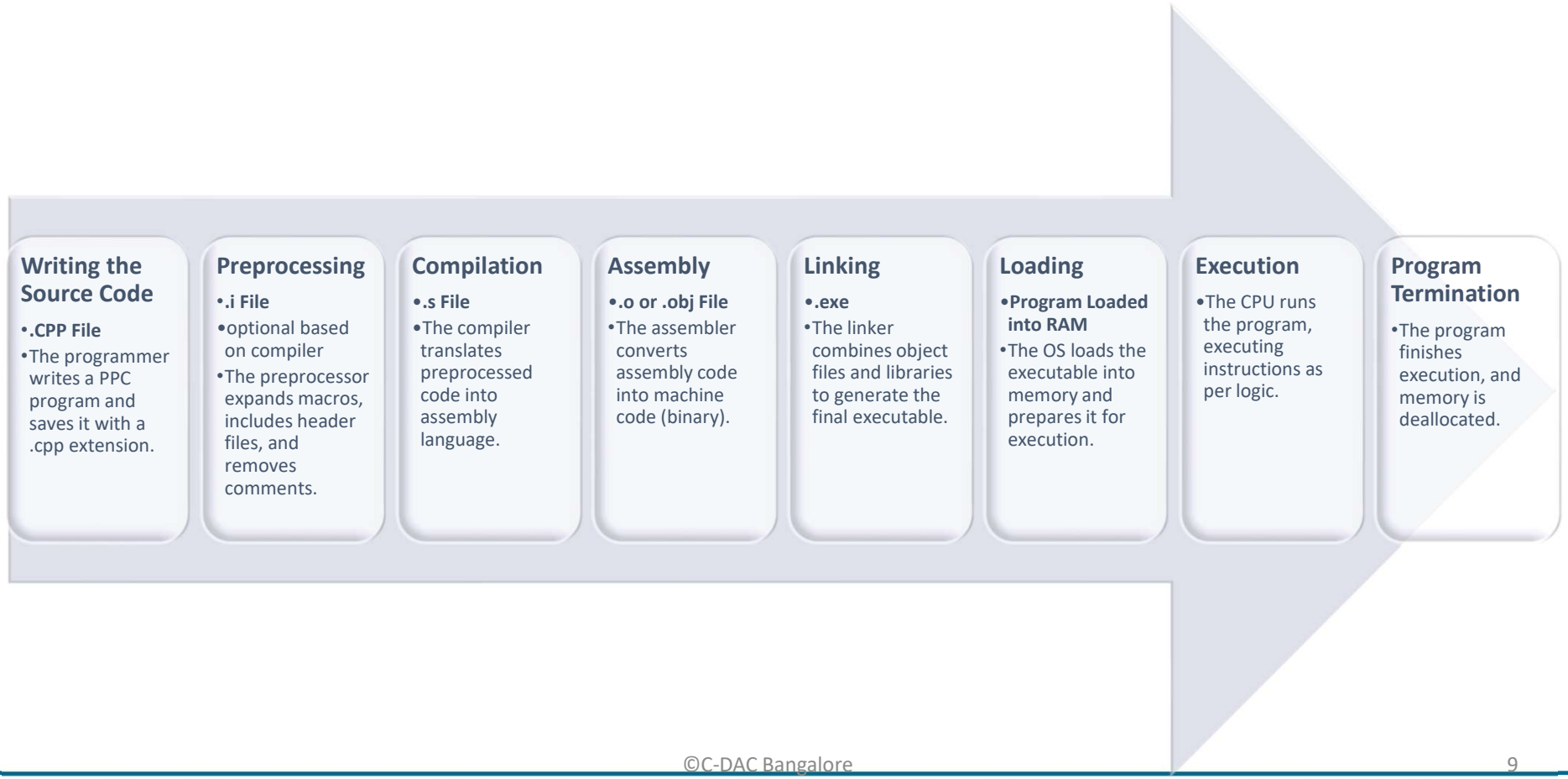
C++23
- Expanded ranges
- Improved constexpr
- std::expected, std::flat_map *(pending compiler support)*

C++ PROGRAM STRUCTURE

Preprocessor Directives	#include <stdio.h> and other header files
Macro Definitions (#define)	Define constants or macros before compilation
	Example: #define PI 3.1415
	Example: #define SQUARE(x) (x * x)
Function & Class Prototypes (Optional)	Declare functions & Class before use for better readability
	Example: int add (int, int);
Global Variable Declarations	Declare variables used across functions
main() Function (Entry Point)	Declare and initialize variables
	Execute logic, function calls, and statements
	Return exit status using return 0;
Function Definitions (User-Defined)	Define functions separately for modular programming



C++ PROGRAM EXECUTE CYCLE



First C++ Program

- Program to display text “HelloWorld on Console
- cout Is used for display on standard output device (Console)
- Namespace std is used for directly utilizing cout from std namespace

```
#include<iostream>

using namespace std;

int main(){
    cout<<"Hello World";
    return 0;
}
```

C++ TOKENS

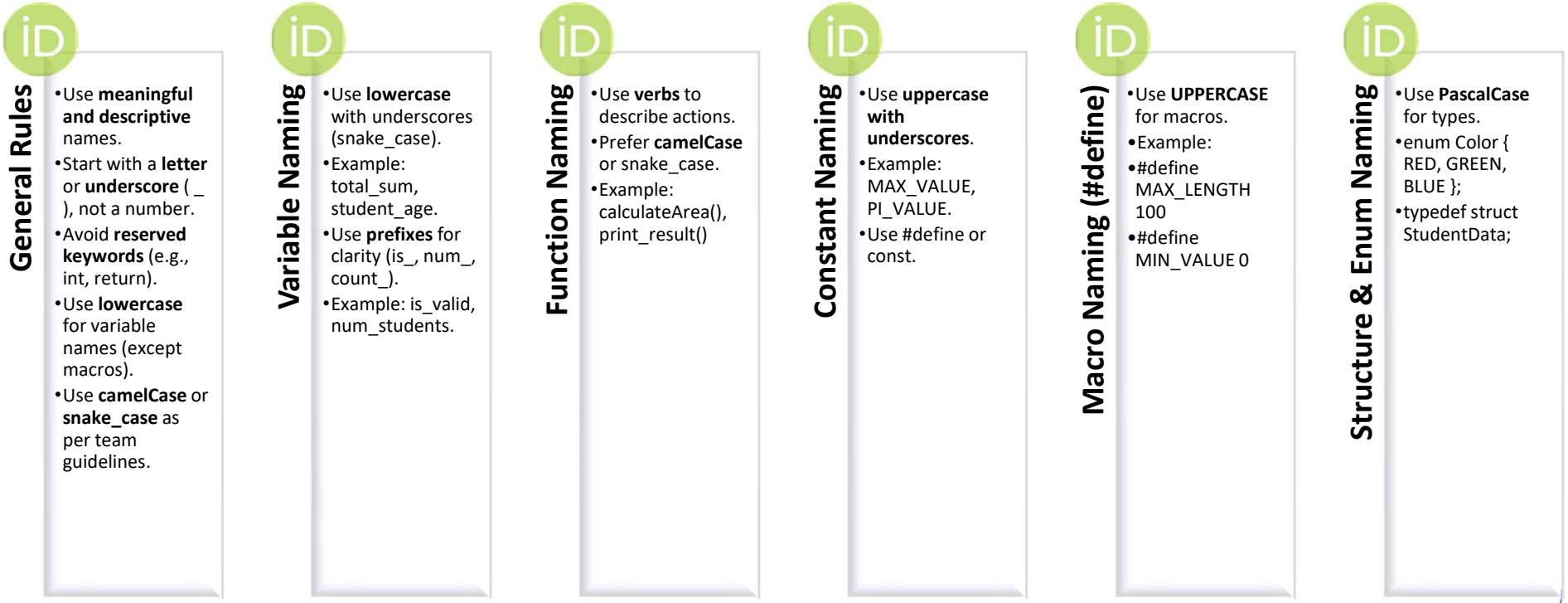
Keyword	Identifier	Literals	String	Operator	Punctuators
Reserved words with predefined meanings	User-defined names for variables, functions, and arrays	Fixed values that do not change during execution	Sequence of characters enclosed in double quotes	symbols that perform operations on variables and values	Characters with special meanings in C
if for include	Sum Result addNumbers	3.142 'A' 0	"PROGRAM" "FALSE"	+ & <	{ . ;

KEYWORDS

C++98 Standard Keywords

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

IDENTIFIERS



LITERALS / CONSTANTS

Integer Literals

Whole numbers (positive, negative, or zero).

Can be **decimal (base 10), octal (base 8), or hexadecimal (base 16)**.

EG:

- int decimal = 100; // Decimal
- int octal = 012; // Octal (starts with 0)
- int hex = 0xA1; // Hexadecimal (starts with 0x)

Floating-Point (Real) Literals

Numbers with a **decimal point** or **exponential notation**.

Example:

- float num1 = 3.14; // Decimal notation
- double num2 = 2.5e3; // Exponential notation (2.5×10^3)

Character Literals

A **single** character enclosed in **single quotes (‘’)**.

Example:

- char letter = 'A';
- char symbol = '#';

String Literals

A **sequence of characters** enclosed in **double quotes ("")**.

Example:

- char name[] = "Hello";

Boolean Literals (C99 & Later)

Represents **true (1) or false (0)**.

Requires <stdbool.h>.

Example:

- #include <stdbool.h>
- bool isValid = true;
- bool isComplete = false;

Escape Sequence Literals

Special characters represented using backslash (\) notation.

Example

- printf("Hello\tWorld!\n"); // \t (Tab), \n (Newline)

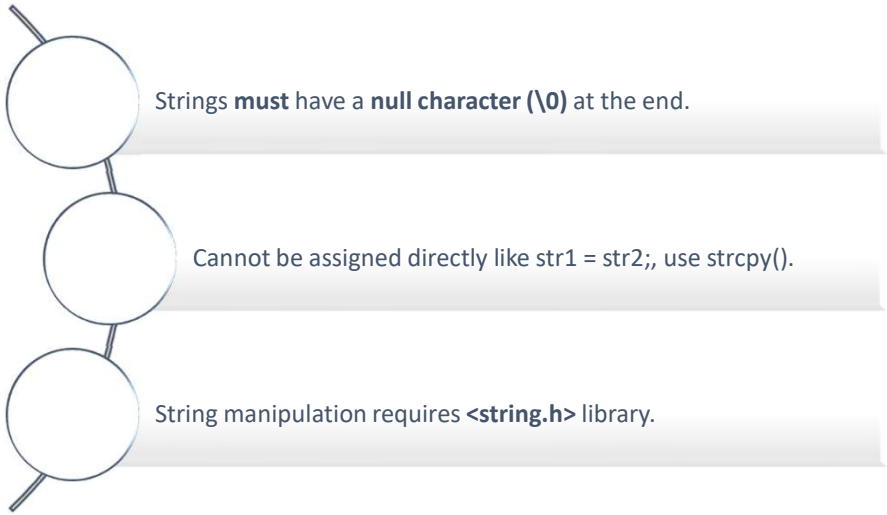
STRINGS

- **sequence of characters** stored as an **array of characters** and **terminated by a null character '\0'**

- **Declaring Strings**

Using Character Array	Using Character Pointer	Explicit Null-Termination
char str[] = "Hello"; // Automatically adds '\0' at the end	char *str = "Hello"; // Stored in read-only memory	char str[] = {'H', 'e', 'l', 'l', 'o', '\0'};

- **Important Points about String**



OPERATORS

Arithmetic Operators (Perform mathematical operations)

- + (Addition) → a + b
- - (Subtraction) → a - b
- * (Multiplication) → a * b
- / (Division) → a / b
- % (Modulus) → a % b (Remainder)

Relational (Comparison) Operators (Compare values)

- == (Equal to) → a == b
- != (Not equal to) → a != b
- > (Greater than) → a > b
- < (Less than) → a < b
- >= (Greater than or equal to) → a >= b
- <= (Less than or equal to) → a <= b

Logical Operators (Used for Boolean logic)

- && (Logical AND) → (a > 0 && b > 0)
- || (Logical OR) → (a > 0 || b > 0)
- ! (Logical NOT) → !(a > b)

Bitwise Operators (Operate on binary values)

- & (Bitwise AND) → a & b
- | (Bitwise OR) → a | b
- ^ (Bitwise XOR) → a ^ b
- ~ (Bitwise NOT) → ~a
- << (Left shift) → a << 2
- >> (Right shift) → a >> 2

Assignment Operators (Assign values to variables)

- = (Assign) → a = 5
- += (Add & assign) → a += 2 (a = a + 2)
- -= (Subtract & assign) → a -= 2
- *= (Multiply & assign) → a *= 2
- /= (Divide & assign) → a /= 2
- %= (Modulus & assign) → a %= 2

Increment & Decrement Operators (Increase or decrease values)

- ++ (Increment) → a++ or ++a
- -- (Decrement) → a-- or –a

Ternary Operator (Conditional Operator)

- condition ? expr1 : expr2

Special Operators

- sizeof (Finds size of variable/type) → sizeof(int)
- & (Address-of) → &var
- * (Pointer dereference) → *ptr
- , (Comma operator) → a = (b = 2, b + 3);

OPERATOR PRECEDENCE

Precedence	Operators	Associativity	Example
1 (Highest)	() [] -> .	**Left to Right**	array[i], ptr->val
2	++ -- (Postfix)	**Left to Right**	x++, y--
3	++ -- (Prefix), + - (Unary), ! ~	**Right to Left**	++x, -y, !flag
4	* / %	**Left to Right**	a * b, x / y
5	+ - (Binary)	**Left to Right**	a + b, x - y
6	<< >>	**Left to Right**	x << 2, y >> 1
7	< <= > >=	**Left to Right**	a < b, x >= y
8	== !=	**Left to Right**	a == b, x != y
9	& (Bitwise AND)	**Left to Right**	a & b
10	^ (Bitwise XOR)	**Left to Right**	a ^ b
11	` (Bitwise OR)	**Left to Right**	a ` b
12	&& (Logical AND)	**Left to Right**	a && b
13	||(Logical OR)	**Left to Right**	a || b
14	?: (Ternary)	**Right to Left**	x = (a > b) ? a : b;
15	= += -= *= /= %=	**Right to Left**	x += 5, y = a * b
16 (Lowest)	, (Comma)	**Left to Right**	a = 1, b = 2

SHORTHAND OPERATORS

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assigns the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C

Variable Storage Types

Automatic

- Default storage class for local variables
- **Scope:** Local
- **Default Value:** Garbage Value
- **Memory Location:** RAM
- **Lifetime:** Till the end of its scope

Static

- **Scope:** Local
- **Default Value:** Zero
- **Memory Location:** RAM
- **Lifetime:** Till the end of the program

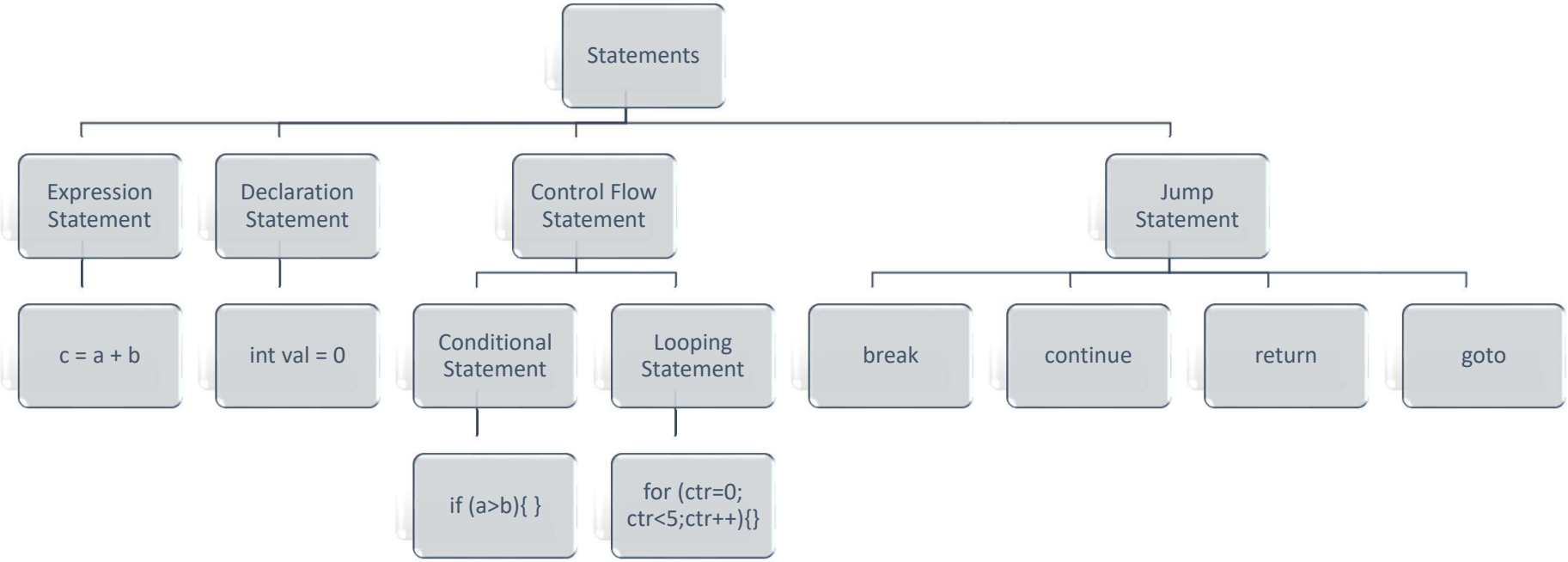
Extern

- **Scope:** Global
- **Default Value:** Zero
- **Memory Location:** RAM
- **Lifetime:** Till the end of the program.

Register

- **Scope:** Local
- **Default Value:** Garbage Value
- **Memory Location:** Register in CPU or RAM
- **Lifetime:** Till the end of its scope

STATEMENTS



CONDITIONAL STATEMENTS

if

```
if (x > 0) {
    cout << "Positive number";
}
```

if else

```
if (x % 2 == 0) {
    cout << "Even";
} else {
    cout << "Odd";
}
```

nested if else

```
if (x > 0) {
    if (x < 10) {
        cout << "Single-digit positive";
    }
}
```

if else ladder

```
if (score >= 90) {
    cout << "Grade A";
} else if (score >= 75) {
    cout << "Grade B";
} else {
    cout << "Grade C";
}
```

switch case

```
int day = 3;
switch(day) {
    case 1: cout << "Monday"; break;
    case 2: cout << "Tuesday"; break;
    case 3: cout << "Wednesday"; break
    default: cout << "Invalid day";
}
```

ternary operator

```
int a = 10, b = 20;
int max = (a > b) ? a : b;
cout << "Maximum is " << max;
```

LOOPING STATEMENTS

```
for (int i = 1; i <= 5; i++) {
    printf("%d ", i);
}
```

For Loop

```
int i = 1;
while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

While Loop

```
int i = 1;
do {
    printf("%d ", i);
    i++;
} while (i <= 5);
```

Do while loop

FOR LOOP

Syntax:

```
for (initialization; condition; update) {
    // Code to execute in each iteration
}
```

Components:

- Initialization → Sets the loop variable (runs once).
- Condition → Checks before each iteration (loop runs if true).
- Iteration/Update → Modifies the loop variable after each iteration.

Example:

```
for (int i = 1; i <= 5; i++) {
    cout<<i;
}
```

WHILE LOOP

Syntax:

```
while (condition) {
    // Code to execute in each iteration
}
```

Components:

- Condition → Checks before each iteration (loop runs if true).

Example:

```
int i = 1;
while (i <= 5) {
    cout<<i;
    i++;
}
```

DO WHILE LOOP

Syntax:

```
do {
    // Code to execute in each iteration
} while (condition);
```

Components:

- Condition → Checks before each iteration (loop runs if true).

Example:

```
int i = 1;
do {
    cout<<i;
    i++;
} while (i <= 5);
```

JUMP STATEMENTS

- **break**
  Breaks out of innermost loop from location of break
- **continue**
  Continues to next iteration of innermost loop to continue
- **return**
  Returns a value from a function
- **goto**
  Goes to a statement with a specific label