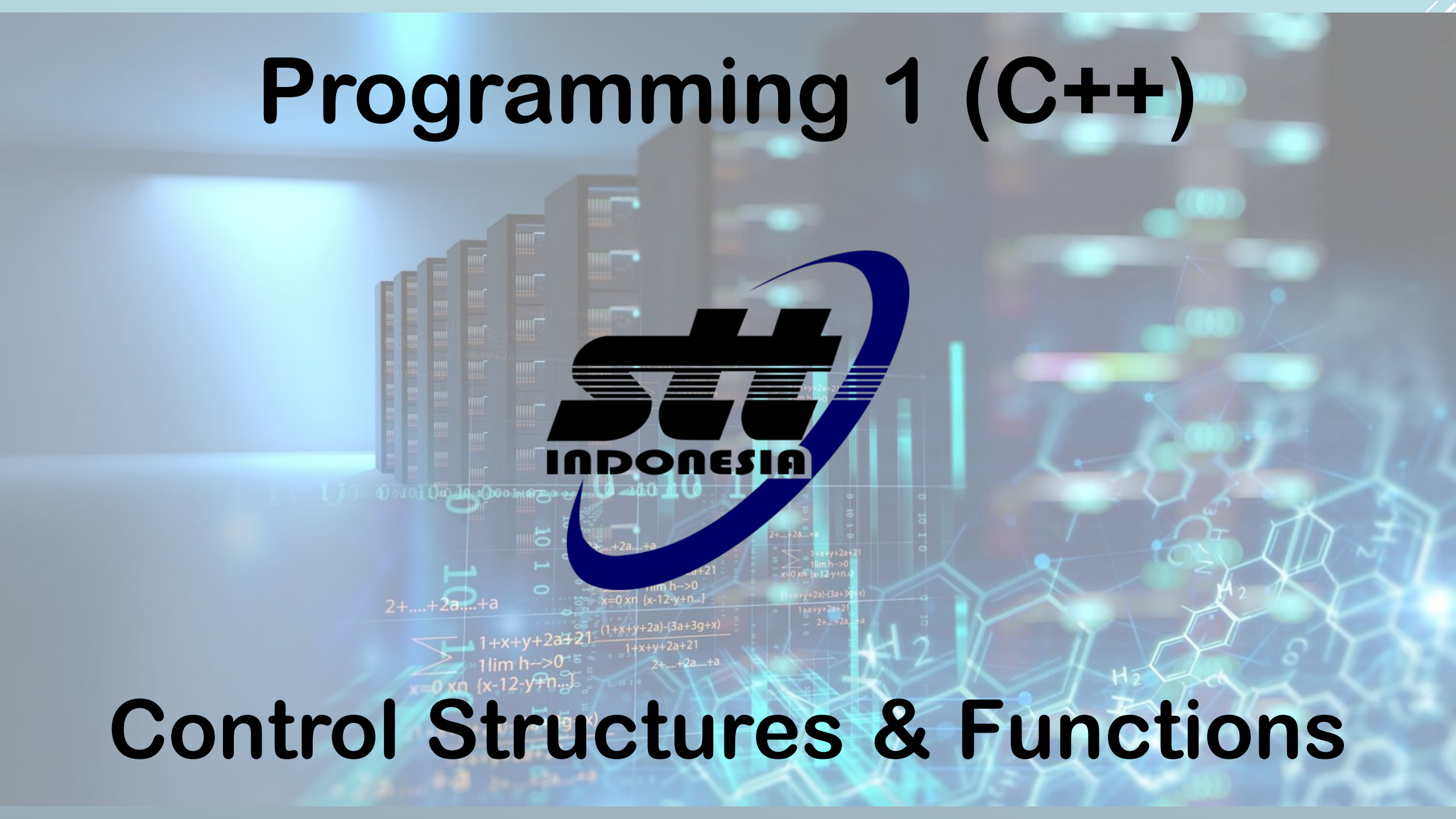
Programming 1 (C++)



Control Structures & Functions

CONTROL FLOW STATEMENT

- Conditional Statements (if, else if, else):
  - “If” Statement :
  - “If-else”
  - “else-if”

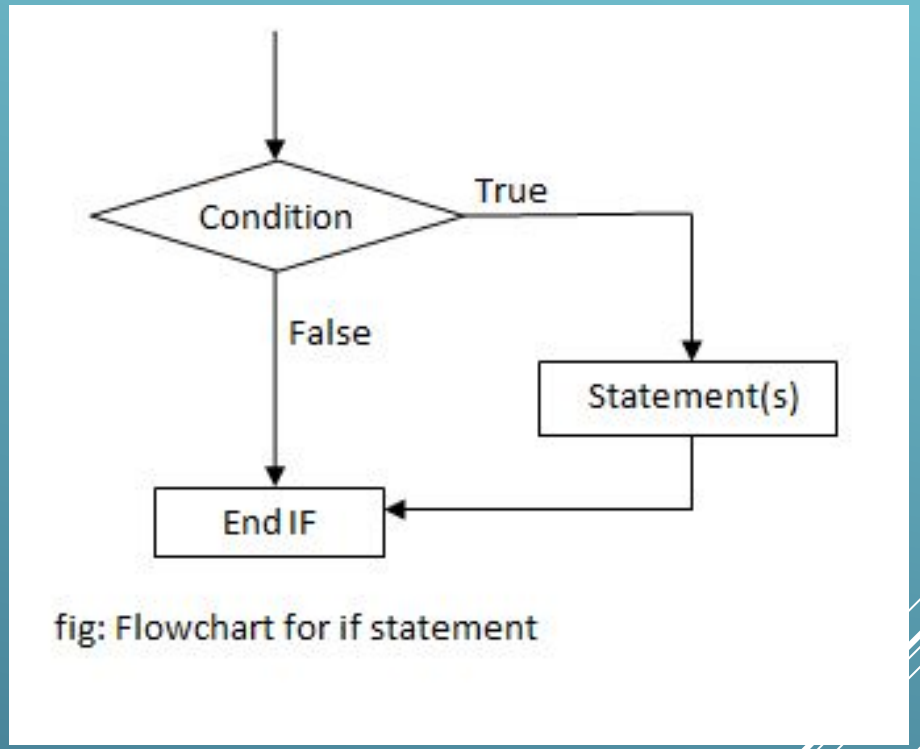


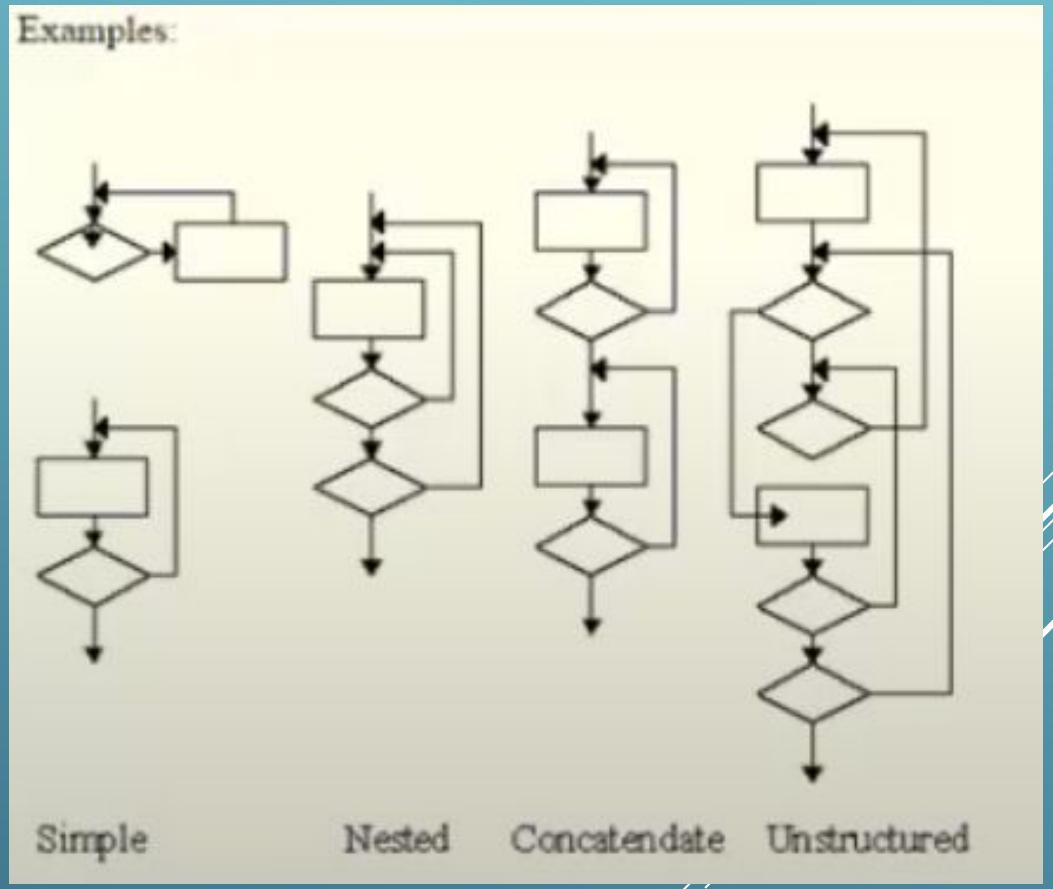
fig: Flowchart for if statement

```
if (condition1) {
    // code to be executed if condition1 is true
} else if (condition2) {
    // code to be executed if condition2 is true
} else {
    // code to be executed if none of the conditions are true
}
```

CONTROL FLOW STATEMENT

- Iterations Statements (loops):
  - “while” loops
  - “do while” loops
  - “ for”

```
while (condition) {
    // code to be executed repeatedly while
condition is true
}

do {
    // code to be executed repeatedly
} while (condition);

for (initialization; condition; update) {
    // code to be executed repeatedly while
condition is true
}
```



CONTROL FLOW STATEMENT

- Jump statement :
  - “break”
  - “continue”
  - “return”

```
if (i == 5) {
            break; // exit the loop when i equals 5
        }

if (i % 2 == 0) {
            continue; // skip even numbers

int result = a + b;
    return result; // return the sum of a and b
```

SWITCH CASE STATEMENT

- Execute different blocks of code based on the value of a variable or an expression

```
switch (expression) {
    case value1:
        // code to be executed if expression equals value1
        break;
    case value2:
        // code to be executed if expression equals value2
        break;
    // additional cases...
    default:
        // code to be executed if expression does not match any case
}
```

FUNCTION AND MODULAR PROGRAMMING

Functions and Their Importance:

- Functions are essential in programming as they allow for the organization and reuse of code.
- They encapsulate a set of instructions to perform a specific task, promoting modularity and readability.
- Functions also facilitate code maintenance and debugging by isolating different functionalities.

Function Parameters and Return Values:

- Function parameters are variables that are passed into a function to provide it with necessary data.
- Return values are the data that a function sends back to the caller after its execution.
- Parameters allow functions to be more versatile by accepting different input values, while return values enable functions to provide output.

Scope and Lifetime of Variables:

- Scope refers to the visibility and accessibility of variables within a program.
- Local variables are declared within a function and are only accessible within that function's block.
- Global variables are declared outside of any function and are accessible throughout the entire program.
- Lifetime refers to the duration for which a variable exists in memory, which is typically determined by its scope.

LOCAL AND GLOBAL VARIABLE

```
// Variabel global
int globalVar = 10;

int main() {
    // Variabel lokal
    int localVar = 20;

    std::cout << "Global variable inside main(): " << globalVar <<
std::endl;
    std::cout << "Local variable inside main(): " << localVar <<
std::endl;

    return 0;
}
```

OUTPUT :

```
Global variable inside main(): 10
Local variable inside main(): 20
```

MODULAR PROGRAMMING

Modular programming is a software development approach that divides programs into smaller, manageable parts, known as modules or functions.

Modules or functions are used to accomplish specific tasks separately, and then combined to form an entire program.

Benefits of Modular Programming:

- Division of Tasks: Modular programming allows programs to be divided into smaller modules or functions, making development, maintenance, and understanding of code easier.
- Code Reuse: Modules or functions that have been created can be reused in other programs, thereby saving time and effort in software development.
- Easier Code Reading: Dividing the program into stand-alone modules or functions makes the code easier to read and understand, because each function is only responsible for one task.

STANDARD LIBRARY FUNCTIONS

The Standard C++ Library is a collection of pre-defined functions and other program elements which are accessed through header files.

- <iostream>: Input and output streams.
- <iomanip>: Manipulators for formatted I/O.
- <fstream>: File I/O streams.
- <cmath>: Mathematical functions.
- <string>: String handling.
- <vector>: Dynamic array implementation.
- <array>: Static array implementation.
- <list>: Doubly linked list implementation.
- <map>: Associative array (map) implementation.
- <set>: Associative container with unique keys (set) implementation.

FUNCTIONS OF LIBRARY (CMATH)

Some Functions Defined in the `<cmath>` Header

Function	Description	Example
`acos(x)`	inverse cosine of `x` (in radians)	`acos(0.2)` returns 1.36944
`asin(x)`	inverse sine of `x` (in radians)	`asin(0.2)` returns 0.201358
`atan(x)`	inverse tangent of `x` (in radians)	`atan(0.2)` returns 0.197396
`ceil(x)`	ceiling of `x` (rounds up)	`ceil(3.141593)` returns 4.0
`cos(x)`	cosine of `x` (in radians)	`cos(2)` returns -0.416147
`exp(x)`	exponential of `x` (base e)	`exp(2)` returns 7.38906
`fabs(x)`	absolute value of `x`	`fabs(-2)` returns 2.0
`floor(x)`	floor of `x` (rounds down)	`floor(3.141593)` returns 3.0
`log(x)`	natural logarithm of `x` (base e)	`log(2)` returns 0.693147
`log10(x)`	common logarithm of `x` (base 10)	`log10(2)` returns 0.30103
`pow(x,p)`	`x` to the power `p`	`pow(2,3)` returns 8.0
`sin(x)`	sine of `x` (in radians)	`sin(2)` returns 0.909297
`sqrt(x)`	square root of `x`	`sqrt(2)` returns 1.41421
`tan(x)`	tangent of `x` (in radians)	`tan(2)` returns -2.18504

STANDARD LIBRARY FUNCTIONS

- <algorithm>: Standard algorithms (e.g., sorting, searching).
- <numeric>: Numeric operations.
- <ctime>: Date and time functions.
- <random>: Random number generation.
- <regex>: Regular expressions.
- <utility>: Various utility components.
- <iterator>: Iterator definitions.
- <exception>: Exception handling.
- <memory>: Dynamic memory management.
- <functional>: Function objects.

Some of the Header Files in the Standard C++ Library

Header File	Description
`<cassert>`	Defines the `assert()` function
`<ctype>`	Defines functions to test characters
`<cfloat>`	Defines constants relevant to floats
`<climits>`	Defines the integer limits on your local system
`<cmath>`	Defines mathematical functions
`<cstdio>`	Defines functions for standard input and output
`<cstdlib>`	Defines utility functions
`<cstring>`	Defines functions for processing strings
`<ctime>`	Defines time and date functions

USER DEFINED FUNCTIONS

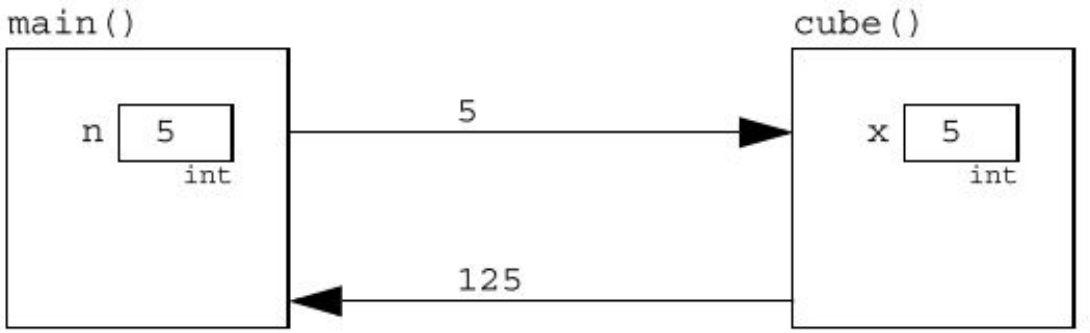
```
int cube(int x)
{ // returns cube of x:
  return x*x*x;
}

int main()
{ // tests the cube() function:
  int n=1;
  while (n != 0)
  { cin >> n;
    cout << "\tcube(" << n << ") = " << cube(n) << endl;
  }
}
```

Head

Body

Test Driver

Relationship



MODULAR PROGRAMMING

```
// Deklarasi fungsi
void printHello() {
    std::cout << "Hello, ";
}

void printWorld() {
    std::cout << "world!" << std::endl;
}

// Fungsi utama
int main() {
    // Panggil fungsi
    printHello();
    printWorld();

    return 0;
}
```

OUTPUT :

```
Hello, world!
```

FUNCTION DECLARATIONS & DEFINITIONS

Functions in C++ are blocks of code that perform a specific task. They provide a way to modularize code, making it easier to read, understand, and maintain.

- Function Prototype (declarations): A function prototype declares the function's name, return type, and parameters, allowing the compiler to recognize the function before it is called.
- Function Definition: The function definition includes the actual implementation of the function, specifying the code that executes when the function is called.

```
// Function prototype
int add(int a, int b);

int main() {
    int result = add(3, 5);
    std::cout << "Result: " << result << std::endl;
    return 0;
}

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

PARAMETERS & ARGUMENTS

PARAMETERS

- The variables that are listed in the function's parameter list are called parameters.
- They are **local variables** that exist only during the execution of the function.

ARGUMENTS

- The variables that are listed in the function's calls are called the arguments.

VOID, BOOLEAN, I/O

Void :

- A void function is simply one that returns no value.
- Example :

```
void printDate(int,int,int);
// // prints the given date in literal form;
```

Boolean :

- a function to evaluate a condition, typically within an if statement or a while statement.
- Example : Classifying Character = isdigit(), islower(), isupper(), isspace(), iscntrl(),and ispunct()

I/O :

- encapsulating tasks that require messy details that are not very relevant to the primary task of the program.
- Example : in processing personnel records.

CALL BY VALUE VS CALL BY REFERENCE

C++ supports two methods of passing arguments to functions:

- Call by Value: In this method, the actual value of the argument is passed to the function. Changes made to the parameter inside the function do not affect the original value of the argument.
- Call by Reference: In this method, the memory address of the argument is passed to the function. Changes made to the parameter inside the function directly affect the original value of the argument.

Passing By Value	Passing By Reference
`int x;`	`int &x;`
The parameter x is a local variable.	The parameter `x` is a local reference.
It is a duplicate of the argument.	It is a synonym for the argument.
It cannot change the argument.	It can change the argument.
The argument passed by value may be a constant, a variable, or an expression.	The argument passed by reference must be a variable.
The argument is read-only.	The argument is read-write.

CALL BY VALUE VS CALL BY REFERENCE

C++ supports two methods of passing arguments to functions:

- Call by Value: In this method, the actual value of the argument is passed to the function. Changes made to the parameter inside the function do not affect the original value of the argument.
- Call by Reference: In this method, the memory address of the argument is passed to the function. Changes made to the parameter inside the function directly affect the original value of the argument.

```cpp
// Fungsi untuk menambahkan nilai x dengan 10
void addTen(int x) {
    x = x + 10; // Menambahkan 10 ke nilai x
    std::cout << "Nilai di dalam fungsi addTen(): " << x << std::endl;
}

int main() {
    int num = 5;
    std::cout << "Nilai sebelum pemanggilan fungsi: " << num << std::endl;
    addTen(num); // Memanggil fungsi addTen() dengan num sebagai argumen
    std::cout << "Nilai setelah pemanggilan fungsi: " << num << std::endl;
    return 0;
}
```

```cpp
// Fungsi untuk menambahkan nilai x dengan 10
void addTen(int &x) {
    x = x + 10; // Menambahkan 10 ke nilai x
    std::cout << "Nilai di dalam fungsi addTen(): " << x << std::endl;
}

int main() {
    int num = 5;
    std::cout << "Nilai sebelum pemanggilan fungsi: " << num << std::endl;
    addTen(num); // Memanggil fungsi addTen() dengan num sebagai argumen
    std::cout << "Nilai setelah pemanggilan fungsi: " << num << std::endl;
    return 0;
}
```

DEFAULT ARGUMENTS

C++ allows you to provide default values for function parameters. If a value is not provided for a parameter during function call, the default value is used.

```cpp
// Function with default argument
void greet(std::string name = "World") {
    std::cout << "Hello, " << name << "!" << std::endl;
}

// Function call with default argument
greet(); // Output: Hello, World!
greet("Alice"); // Output: Hello, Alice!
```

INLINE FUNCTIONS

Inline functions are small functions that are expanded at the point of their call rather than being called like a regular function. This can improve performance by eliminating the overhead of a function call.

```
// Inline function
inline int square(int x) {
    return x * x;
}
```

OVERLOADED FUNCTIONS

Function overloading allows you to define ***multiple functions with the same name*** but different parameter lists. The compiler selects the appropriate function to call based on the number and types of arguments provided.

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

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

TUGAS KELOMPOK 2

Variabel Lokal dan Global, Standard Library Function:

Tulis sebuah program C++ yang menunjukkan penggunaan variabel lokal dan global dalam fungsi. Gunakan setidaknya tiga Standard Library Function dalam program Anda. Pastikan Anda dengan jelas menandai mana variabel yang lokal dan mana yang global.

THANK YOU