ASSIGNMENT -2

1. What is the purpose of the main function in a C++ program?

The `main` function is the **entry point** of every C++ program. When the program is executed, execution begins from the `main` function. It serves as the **starting place** where instructions are first carried out. All other functions are called from within `main` (directly or indirectly), making it essential in defining the overall flow of the program.

2. Explain the significance of the return type of the main function

The return type of the `main` function is typically `int`, which means it returns an integer value to the **operating system** after the program completes execution.

Significance:

- Returning `0` usually means the program ended successfully.
- Returning a **non-zero** value often indicates an **error or abnormal termination**.
- It helps in **process control** and is used by the operating system or other programs/scripts to check the status of the program's execution.

3. What are the two valid signatures of the main function in C++?

In C++, the two valid signatures (forms) of the `main` function are:

```
cpp
CopyEdit
int main()  // No command-line arguments
```

and

```
cpp
CopyEdit
int main(int argc, char* argv[])  // With command-line arguments
```

- `argc` is the argument count (number of command-line arguments).
- `argv` is an array of character pointers (the actual arguments).

4. What is function prototyping and why is it necessary in C++?

Function prototyping is the declaration of a function **before** its actual definition, specifying its return type, name, and parameter types.

Why it's necessary:

- It allows the compiler to **check function calls for correctness** (type and number of arguments).
- It **enables calling a function before it is defined**, which is crucial in multi-file programs or when function definitions appear later in the code.

Example:

```
cpp
CopyEdit
int add(int, int);  // Function prototype

int main() {
    int result = add(3, 4);  // Safe to use add here
    return 0;
}

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

5. How do you declare a function prototype for a function that returns an integer and takes two integer parameters?

You declare it like this:

```
cpp
CopyEdit
int functionName(int, int);
```

Example:

```
cpp
```

What is the difference between a declaration and a definition of a function?

- **Function Declaration (Prototype):**
  - Tells the compiler **what the function looks like** (its return type, name, and parameters).
  - Does **not** include the function body.
  - Ends with a semicolon.
  - Example:

    ```
    cpp
    CopyEdit
    void greet();  // Declaration
    ```

- **Function Definition:**
  - Provides the **actual body** of the function (implementation).
  - Contains the code that gets executed when the function is called.
  - Example:

    ```
    cpp
    CopyEdit
    ```

```
void greet() {
    cout << "Hello!";
}  // Definition
```

8. How do you call a simple function that takes no parameters and returns void?

You call it using its name followed by empty parentheses:

```
cpp
CopyEdit
void greet();  // Declaration

int main() {
    greet();  // Function call
    return 0;
}

void greet() {
    cout << "Hello!";
}
```

9. Explain the concept of "scope" in the context of functions.

Scope refers to the **visibility and lifetime** of variables and functions—i.e., where in the program they can be accessed.

- **Global scope:** Variables/functions declared outside of all functions. Accessible from anywhere in the file.
- **Local scope:** Variables declared inside a function. They only exist within that function.
- **Function scope:** Functions themselves have scope; once defined, they are globally accessible (unless declared `static` or inside a class).

Example:

```
cpp
CopyEdit
int x = 10;  // Global variable

void show() {
    int y = 5;  // Local variable
    cout << x + y;
}
```

10. What is call by reference in C++?

Call by reference means passing the **reference (alias)** of a variable to a function, so changes made to the parameter **affect the original variable**.

Syntax:

```
cpp
CopyEdit
void modify(int &num) {
    num = num + 5;  // Modifies the original variable
}

int main() {
    int x = 10;
    modify(x);
    cout << x;  // Outputs 15
}

int sum(int, int);
```

11. How does call by reference differ from call by value?

- **Call by Value:**
  A **copy** of the variable is passed to the function. Changes made inside the function **do not affect** the original variable.
- **Call by Reference:**
  The **actual variable** is passed using a reference (`&`). Changes made inside the function **directly affect** the original variable.

12. Provide an example of a function that uses call by reference to swap two integers.

```
cpp
CopyEdit
#include <iostream>
using namespace std;

void swap(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 5, y = 10;
    swap(x, y);
    cout << "x = " << x << ", y = " << y;
    return 0;
}
```

13. What is an inline function in C++?

An **inline function** is a function where the compiler **replaces the function call with the actual code of the function** at compile time, to eliminate the overhead of a function call.

14. How do inline functions improve performance?

They improve performance by:

- **Avoiding function call overhead** (no jump to a separate location in memory).
- **Speeding up execution** for small, frequently called functions.

However, they can increase the size of the binary if overused (known as **code bloat**).

15. Explain the syntax for declaring an inline function.

You use the `inline` keyword before the function definition:

```
cpp
CopyEdit
inline int square(int x) {
    return x * x;
}
```

16. . What are macros in C++ and how are they different from inline functions?

Macros are defined using the `#define` directive and are handled by the **preprocessor** (before compilation). They perform **text substitution**.

Differences:

Feature	Macros	Inline Functions
Handled by	Preprocessor	Compiler
Type checking	No	Yes
Debugging	Difficult	Easier
Scope awareness	No (global)	Yes
Safer to use	✗	✓

Example of macro:

```
cpp
CopyEdit
#define SQUARE(x) ((x) * (x))
]
```

17. Advantages and Disadvantages of Using Macros over Inline Functions

Advantages:

- Macros can be used for simple constants and small code blocks.
- Slightly faster for very small operations, in rare cases.

Disadvantages:

- No **type safety** or **error checking**.**Conclusion:** Inline functions are generally safer and more reliable than macros.

macros and inline functions.

```
cpp
CopyEdit
#include <iostream>
using namespace std;

#define SQUARE_MACRO(x) ((x) * (x))  // Macro

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

int main() {
    int a = 5;
    int b = SQUARE_MACRO(a + 1);     // Expands to ((a + 1) * (a + 1)) = (5
+ 1 * 5 + 1) = wrong!
    int c = square_inline(a + 1);    // Correct: (6 * 6) = 36

    cout << "Macro result: " << b << endl;
    cout << "Inline result: " << c << endl;

    return 0;
}
```

Output:

```
yaml
CopyEdit
Macro result: 11   (Incorrect due to no parentheses in evaluation)
Inline result: 36  (Correct)
```

Conclusion: Macros perform raw text substitution, which can cause unexpected results. Inline functions are safer due to type checking and proper evaluation.

19. What is function overloading in C++?

Function overloading is when multiple functions have the **same name but different parameters** (number or type). The compiler chooses the right one based on how it's called.

20. How does the compiler differentiate between overloaded functions?

The compiler uses the **function signature**: the **number and type of parameters**. The return type is **not** considered in overload resolution.

21. Provide an example of overloaded functions in C++.

```
cpp
CopyEdit
#include <iostream>
using namespace std;

void print(int i) {
    cout << "Integer: " << i << endl;
}

void print(double d) {
    cout << "Double: " << d << endl;
}

void print(string s) {
    cout << "String: " << s << endl;
}

int main() {
    print(5);          // Calls print(int)
    print(3.14);       // Calls print(double)
    print("Hello");    // Calls print(string)
    return 0;
}
```

22. What are default arguments in C++?

Default arguments are values assigned to function parameters **in the declaration**. If the caller omits some arguments, the default values are used.

23. How do you specify default arguments in a function declaration?

Default arguments are added in the **function declaration**, not necessarily the definition.

```
cpp
CopyEdit
void greet(string name = "Guest");
```

Then in the definition:

```
cpp
CopyEdit
void greet(string name) {
    cout << "Hello, " << name << "!" << endl;
}
```

24. What are the rules for using default arguments in functions?

1. **Once a default argument is provided, all arguments to the right must also have defaults.**

   ```
   cpp
   CopyEdit
   void example(int a = 10, int b = 20);  // ✅ Valid
   void wrong(int a = 10, int b);         // ❌ Invalid
   ```

2. Default arguments should be specified in **either the declaration or the definition**, not both.
3. They are evaluated **at the point of the call**, not at the declaration.

25. Provide an example of a function with default arguments.

✅ Example: Function with Default Arguments

```
cpp
CopyEdit
#include <iostream>
using namespace std;

// Function declaration with default arguments
void greet(string name = "Guest", int times = 1);

int main() {
    greet();                // Uses both default values
    greet("Alice");         // Uses default for 'times'
    greet("Bob", 3);        // No defaults used

    return 0;
}

// Function definition
void greet(string name, int times) {
    for (int i = 0; i < times; ++i) {
        cout << "Hello, " << name << "!" << endl;
    }
}
```

🔍 Output:

```
CopyEdit
Hello, Guest!
Hello, Alice!
Hello, Bob!
Hello, Bob!
```