**Assignment-2**

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

Ans-In a C++ program, the main function serves as the starting point for program execution.

When you run a C++ program, the execution always begins from the main function.  
Its main purposes are:

* It controls the flow of the program by calling other functions if needed.
* It usually returns an integer (commonly 0) to the operating system to indicate whether the program ended successfully or with an error.
* It initializes the program and manages its overall behavior.

Example:

#include <iostream>

using namespace std;

int main() {

cout << "Hello, world!" << endl;

return 0; // 0 indicates successful execution

}

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

Ans-he return type of the main function in C++ is int (integer), and it has an important role:

* It sends a value back to the operating system (or the environment that started the program) after the program finishes running.
* A return value of0 usually means successful execution.
* A non-zero value (like 1, -1, etc.) typically signals that the program ended withanerror or abnormal condition.

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

Ans-Two valid signatures of main() in C++ are:

1. int main()  
   → No command-line arguments.
2. int main(int argc, char\* argv[])  
   → Accepts command-line arguments.

Both must return an int to indicate success (0) or error (non-zero).

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

Ans-Function prototyping in C++ means declaring a function before using it, by specifying its name, return type,and parameters — but without writing the full body.

Example of function prototype

int add(int a, int b); // This is a prototype

int main() {

int result = add(5, 3);

cout << result;

}

int add(int a, int b) { // Full definition later

return a + b;

}

Why is it necessary?

* It informs the compiler about the function’s name, return type, and parameters before it is actually used.
* It avoids errors during compilation (otherwise the compiler doesn't know about the function).
* It allows flexible ordering of functions (you can call a function before fully defining it).

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

Ans-You declare it like this:

cpp

int functionName(int a, int b);

* int → return type (function will return an integer)
* functionName → name of the function (you can choose any name)
* (int a, int b) → two integer parameters

**Example:**

cpp

int add(int x, int y);

This prototype says:  
*"*There is a function called add that takes two integers and returns an integer.

6. What happens if a function is used before it is prototyped?

Ans-If a function is used before it is prototyped in C++, the compiler gets confused because it doesn’t know:

* the function’s return type,
* the function’s parameters,
* or even whether the function exists.

As a result:

* You will get a compilation error like "function not declared**"**.
* Or the compiler might assume wrong types, leading to unexpected behavior (especially in older C++ versions).

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

Ans-

| **Declaration** | **Definition** |
| --- | --- |
| Tells the compiler what the function looks like (return type, name, parameters). | Actually provides the full code (body) of the function. |
| No body, only ends with a semicolon (;). | Has thefull body enclosed in {}. |
| Example: int add(int a, int b); | Example:  int add(int a, int b) { return a + b; } |

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

Ans-To call a simple function that takes no parameters and returns void, you just use the function name followed by empty parentheses () and no return value.

Example:

Cpp

#include <iostream>

using namespace std;

// Function definition

void displayMessage() {

cout << "Hello, this is a simple function!" << endl;

}

int main() {

// Function call

displayMessage(); // No parameters, no return value

return 0;

}

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

Ans-In C++, scope refers to the region of the program where a particular variable or function is accessible or visible. The scope of a function determines where the function can be called and where its parameters and local variables can be accessed.

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

Ans-Call by reference in C++ means that when you pass an argument to a function, you pass the memory address (or reference) of the actual variable, rather than a copy of its value.

Key Characteristics:

* The function modifies the original variable (not a copy).
* You use the reference operator & in the function's parameter list to indicate a reference.

**Syntax:**

cpp

void functionName(int& param) {

param = param + 10; // Modifies the original variable

}

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

Ans-**Call by Reference** and **Call by Value** are two different ways of passing arguments to functions in programming. Here's how they differ:

**Call by Value**

* **What it means:** A **copy** of the actual value is passed to the function.
* **Effect:** Changes made inside the function **do not affect** the original variable.
* **Used in:** Languages like **Java (for primitives)**, **C**, **Python (for immutable types)**.

**Example in C:**

cpp

void update(int x) {

x = x + 5;

}

int main() {

int a = 10;

update(a);

// a is still 10

}

**Call by Reference**

* **What it means:** The **address (reference)** of the actual variable is passed to the function.
* **Effect:** Changes made inside the function **do affect** the original variable.
* **Used in:** Languages like **C++**, **Python (for mutable types)**.

**Example in C++:**

cpp

void update(int &x) {

x = x + 5;

}

int main() {

int a = 10;

update(a);

// a becomes 15

}

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

Ans-#include <iostream>

using namespace std;

// Function to swap values using call by reference

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

int main() {

int x = 10, y = 20;

cout << "Before swap: x = " << x << ", y = " << y << endl;

swap(x, y); // Call by reference

cout << "After swap: x = " << x << ", y = " << y << endl;

return 0;

}

Output-

Before swap: x = 10, y = 20

After swap: x = 20, y = 10

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

Ans-An **inline function** in C++ is a function where the **compiler replaces the function call with the actual code of the function** — directly at the place where it's called. This helps to **reduce the overhead of function calls**, especially for small, frequently used functions.

**Key Points:**

* Declared using the inline keyword.
* Suitable for **small** and **simple** functions.
* Improves performance by avoiding function call overhead.
* The compiler **may ignore** the inline request if it decides it's not optimal.

cpp

#include <iostream>

using namespace std;

inline int square(int x) {

return x \* x;

}

int main() {

int num = 5;

cout << "Square: " << square(num) << endl; // Replaced directly with num\*num

return 0;

}

14. How do inline functions improve performance?

Ans-**Inline functions improve performance** by eliminating the **function call overhead**. Here's how:

**What is Function Call Overhead?**

When a normal (non-inline) function is called, the system performs several steps:

1. Pushes arguments onto the stack.
2. Jumps to the function's memory location.
3. Executes the function.
4. Returns back to the calling location.

This process takes time — especially if the function is small and called frequently.

**How Inline Functions Help:**

* The **code of the function is directly inserted** (or "inlined") at the call site.
* This avoids:
  + Jumping to a new location.
  + Stack operations.
  + Return steps.
* As a result, execution is **faster**, especially in **tight loops** or **performance-critical code**.

**Example:**

cpp

inline int add(int a, int b) {

return a + b;

}

int main() {

int result = add(5, 3); // Becomes: int result = 5 + 3;

}

15. Explain the syntax for declaring an inline function.

Ans- syntax-

inline return\_type function\_name(parameter\_list) {

// function body

}

Example-

#include <iostream>

using namespace std;

// Declaring an inline function

inline int square(int x) {

return x \* x;

}

int main() {

int num = 4;

cout << "Square of " << num << " is " << square(num) << endl;

Bottom of Form

return 0;

}

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

Ans-**Macros** are **preprocessor directives** that define constant values or reusable code snippets using the #define keyword. They are handled by the **preprocessor**, **before** actual compilation.

**Syntax:**

**#define macro\_name replacement\_code**

**Example**

**#define SQUARE(x) ((x) \* (x))**

**int main() {**

**int a = 5;**

**int result = SQUARE(a); // Becomes ((5) \* (5))**

**}**

**Difference Between Macros and Inline Functions**

| **Feature** | **Macros (#define)** | **Inline Functions (inline)** |
| --- | --- | --- |
| **Processed by** | Preprocessor (text substitution) | Compiler (type-checked code) |
| **Syntax checking** | No type checking | Fully type-checked |
| **Debugging** | Hard to debug (no symbol) | Easier to debug |
| **Scope rules** | No scope | Respects C++ scope rules |
| **Error-prone?** | Yes (e.g., operator issues) | Safer and cleaner |
| **Example issue** | SQUARE(x + 1) → wrong result | square(x + 1) → correct result |

17. Explain the advantages and disadvantages of using macros over inline functions.

Ans-**Advantages of Macros over Inline Functions:**

| **Advantage** | **Explanation** |
| --- | --- |
| **No function call overhead** | Like inline functions, macros are replaced directly in code (text substitution), so no runtime cost. |
| **Can be used with all types** | Since macros are just text replacement, they work without worrying about data types. |
| **Available before compilation** | Macros are processed by the preprocessor, allowing conditional compilation and other preprocessing tricks. |

**Disadvantages of Macros Compared to Inline Functions:**

| **Disadvantage** | **Explanation** |
| --- | --- |
| **No type checking** | Macros don’t check data types, which can lead to runtime bugs. |
| **No scope rules** | Macros don’t follow C++ scope; they are globally replaced, which can cause naming conflicts. |
| **Debugging is harder** | Since macros don’t exist at runtime, debuggers can’t trace them easily. |
| **Error-prone** | Macros with parameters can behave unexpectedly due to operator precedence (e.g., #define SQUARE(x) x\*x). |
| **No overloading** | You can’t overload macros like you can with inline functions in C++. |

18. Provide an example to illustrate the differences between macros and inline functions.

Ans-

cpp

#include <iostream>

using namespace std;

#define SQUARE(x) (x \* x) // Macro definition

int main() {

int a = 5;

int b = SQUARE(a + 1); // Expands to: (a + 1 \* a + 1)

cout << "Result using macro: " << b << endl;

return 0;

}

Output-

Result using macro: 11

#include <iostream>

using namespace std;

inline int square(int x) {

return x \* x;

}

int main() {

int a = 5;

int b = square(a + 1); // Evaluated properly with correct precedence

cout << "Result using inline function: " << b << endl;

return 0;

}

Output

Result using inline function: 36

19.What is the function overloading in c++?

Ans-**Function overloading** in C++ means having **multiple functions with the same name** but **different parameters** (number, type, or order). The compiler decides **which version** of the function to call based on the **arguments** provided.

Example-

cpp

#include <iostream>

using namespace std;

// Function with one int parameter

void display(int a) {

cout << "Integer: " << a << endl;

}

// Function with one double parameter

void display(double d) {

cout << "Double: " << d << endl;

}

// Function with two parameters

void display(int a, int b) {

cout << "Two integers: " << a << " and " << b << endl;

}

int main() {

display(10); // Calls display(int)

display(5.5); // Calls display(double)

display(2, 3); // Calls display(int, int)

return 0;

}

Output

Integer: 10

Double: 5.5

Two integers: 2 and 3

20. How does the compiler differentiate between overloaded functions?

Ans-In C++, the compiler differentiates overloaded functions based on their **parameter types**, **number of parameters**, and **order of parameters**. The **function name** remains the same, but the **signature** (name + parameters) must be different.

**Key Points:**

* The compiler looks at the function's **parameters** (type, number, and order).
* **Return type** cannot be used to differentiate overloaded functions.

**Example:**

**Cpp**

void print(int x); // One int

void print(double x); // One double

void print(int x, int y); // Two ints

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

Ans-#include <iostream>

using namespace std;

// Overloaded function to print an integer

void print(int x) {

cout << "Integer: " << x << endl;

}

// Overloaded function to print a double

void print(double x) {

cout << "Double: " << x << endl;

}

// Overloaded function to print two integers

void print(int x, int y) {

cout << "Two integers: " << x << " and " << y << endl;

}

int main() {

print(5); // Calls print(int)

print(3.14); // Calls print(double)

print(10, 20); // Calls print(int, int)

return 0;

}

Output-

Integer: 5

Double: 3.14

Two integers: 10 and 20

22. What are default arguments in C++?

Ans-**Default arguments** in C++ allow you to specify default values for function parameters. If the caller doesn't provide a value for a parameter, the default value is used.

**Syntax:**

cpp

return\_type function\_name(parameter1 = default\_value1, parameter2 = default\_value2, ...);

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

Ans-In C++, **default arguments** are specified in the **function declaration** (or prototype), not in the function definition. You provide default values for parameters in the function declaration, so if a caller does not pass an argument for a parameter, the default value is used.

**Syntax for Declaring Default Arguments:**

cpp

return\_type function\_name(parameter1 = default\_value1, parameter2 = default\_value2, ...);

**Example:**

cpp

#include <iostream>

using namespace std;

// Function declaration with default arguments

void greet(string name, string message = "Hello");

int main() {

greet("John"); // Uses default message "Hello"

greet("Alice", "Good Morning"); // Uses custom message "Good Morning"

return 0;

}

// Function definition (can be without default values)

void greet(string name, string message) {

cout << message << ", " << name << "!" << endl;

}

**Output:**

CopyEdit

Hello, John!

Good Morning, Alice!

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

Ans-Here are the rules for using default arguments in C++ briefly:

1.Default arguments are specified in the function declaration (prototype), not in the definition.

cpp

void greet(string name, string message = "Hello"); // Declaration with default

void greet(string name, string message) { // Definition without default

// ...

}

**2.**Defaults must be assigned from right to left. You can't skip a middle parameter.

cpp

void greet(string name, string message = "Hello", int times = 1); // Valid

**3.**Caller can omit the argument, and the default value will be used.

cpp

greet("John"); // Uses default "Hello"

greet("Alice", "Good Morning"); // Uses custom message

**4.**Don't repeat default values in both declaration and definition.

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

Ans-Example-

#include <iostream>

using namespace std;

// Function declaration with default arguments

void greet(string name, string message = "Hello", int times = 1);

int main() {

greet("John"); // Uses default message "Hello" and times = 1

greet("Alice", "Good Morning"); // Uses custom message "Good Morning" and times = 1

greet("Bob", "Hi", 3); // Uses custom message "Hi" and times = 3

return 0;

}

// Function definition without default values

void greet(string name, string message, int times) {

for (int i = 0; i < times; i++) {

cout << message << ", " << name << "!" << endl;

}

}

Output

Hello, John!

Good Morning, Alice!

Hi, Bob!

Hi, Bob!

Hi, Bob!

Bottom of Form

Top of Form

Bottom of Form