1.What is an **array** in C++?

a) A data type to store single values

b) A collection of elements of the same data type

c) A collection of elements of different data types

d) A mathematical expression

2.How do you **declare** an array in C++?

a) int array b) array<int>

c) int[] array d) int array[ ]

3.What is a **pointer** in C++?

a) A variable that stores the memory address of another variable

b) A variable that stores the value of another variable

c) A variable that stores the result of a mathematical operation

d) A variable that stores a string of characters

4.How do you **declare** a pointer to an integer in C++?

a) int ptr; b) pointer int; c) int\* ptr; d) int& ptr;

5.Which operator is used to access the **value pointed to by a pointer** in C++?

a) & b) \* c) -> d) ::

6.What is "**pointer arithmetic**" in C++?

a) Performing mathematical operations using pointers.

b) A method for allocating memory for pointers.

c) The process of creating new pointers.

d) A way to access array elements using pointers.

7.What is the **difference** between an **array** and a **pointer** in C++?

a) Arrays can store multiple values, while pointers can only store one value.

b) Arrays cannot be used with loops, while pointers can.

c) Arrays are fixed in size, while pointers can be dynamically allocated.

d) Arrays and pointers are essentially the same in C++.

8.Choose one of the best options. An element Referring outside array bounds is a \_\_\_?

A. Logical error B. Syntax error  
C. Execution time error D. Both A and C

9.A **binary search algorithm** is an algorithm that is used for

A. Divide and conquer method B. Linear way to search values  
C. Bubble sorting technique D. None of them

10.**Consecutive group** of memory locations contains all **same name and same type**, is called as

A. Structures B. Arrays C. Classes D. Functions

11.Which of the header file is used for **array type manipulation**?

a) <array> b) <type\_traits>

c) <iostream> d) std namespace

12.What is a "**pointer to a pointer**" in C++?

a) A pointer that cannot be dereferenced.

b) A pointer that points to a function.

c) A pointer that stores the address of another pointer.

d) A pointer that points to an integer value.

13.How do you **declare** a pointer to a pointer to an integer in C++?

a) int\*\* ptr; b) int\* ptr; c) int& ptr; d) pointer\*\* ptr;

14.What is the **primary use of a pointer to a pointer** in C++?

a) To create multi-dimensional arrays. b) To create dynamic data structures.

c) To perform arithmetic operations. d) To access class members.

15>What is the result of the following code?

int x = 10;

int\* ptr1 = &x;

int\*\* ptr2 = &ptr1;

std::cout << \*\*ptr2;

a) 10 b) &x c) 0 d) Error

16.What is the relationship between a pointer to a pointer and a two-dimensional array in C++?

a) They are the same thing and can be used interchangeably.

b) A pointer to a pointer is used to represent a single element in a two-dimensional array.

c) A pointer to a pointer can be used to create and manipulate a two-dimensional array.

d) A two-dimensional array cannot be represented using pointers to pointers.

17.What is the **size of a pointer variable** in C++ on a **32-bit** system?

a.4 byte b. 8 byte c.16 byte d.32 byte

18.What happens if you **dereference a null pointer** in C++?

a.It returns 0 b.It generates a runtime error

c.It returns garbage data d. none

19.What is a **function** in C++?

A. A class member that stores data

B. A block of code that performs a specific task

C. A variable that holds the return value

D. A header file for input/output operations

20.What is the purpose of the **"void"** keyword in a function declaration?

A. It specifies that the function doesn't return a value.

B. It defines a variable within the function.

C. It specifies that the function returns an integer.

D. It makes the function a constructor.

21. What is a **recursive function** in C++?

A. A function that is called from another function.

B. A function that has no return statement.

C. A function that calls itself to solve a problem. D. A function that returns a random value.

22. What is **function overloading** in C++?

A. It is a technique to define multiple functions with the same name in a program.

B. It is a way to limit the number of functions in a program.

C. It is a method to call functions from one program to another.

D. It is a way to define functions with different return types.

23.When can you use **function overloading** in C++?

A. When the functions have the same name and the same number of parameters.

B. When the functions have the same name but different parameter types or a different number of parameters.

C. When the functions have different names.

D. When the functions have the same name but are defined in different source files.

24.Which of the following is NOT a valid condition for function overloading in C++?

A. Different number of parameters B. Different order of parameters

C. Different return type D. Different parameter types

25>If you define two functions with the same name but different parameter types, which function will be called when you pass an argument to the function?

A. The first function defined in the program.

B. The second function defined in the program.

C. The function that matches the argument type most closely.

D. The function with the largest number of parameters.

26.In C++, can you overload functions by changing the order of the parameters?

A. Yes, as long as the types of the parameters are the same.

B. Yes, the order of parameters doesn't matter in function overloading.

C. No, changing the order of parameters is not allowed for function overloading.

D. Yes, but only for non-member functions.

27. What is **dynamic memory allocation** in C++?

A. Allocating memory on the stack at compile time

B. Allocating memory on the heap at compile time

C. Allocating memory on the stack at runtime

D. Allocating memory on the heap at runtime

28.Which C++ operator is used **to allocate memory on the heap**?

A. new B. malloc C. alloc D. heap

29. What is the **purpose of dynamic memory allocation**?

A. To allocate memory for global variables

B. To allocate memory for local variables

C. To allocate memory for variables with automatic storage duration

D. To allocate memory for data structures at runtime

30. In C++, what does the new operator return when it fails to allocate memory?

A. nullptr B. 0 C. An exception is thrown D. A system error code

31.How is **memory deallocated when using dynamic memory allocation** in C++?

A. Automatically by the C++ runtime system

B. By the operating system

C. Using the delete operator or delete[] operator

D. Automatically when the program exits

32.What is the **purpose of the delete operator** in C++?

A. To free memory allocated on the stack

B. To free memory allocated on the heap

C. To release memory used by global variables

D. To release memory used by local variables

33.What is a **potential issue with using dynamic memory allocation** in C++?

A. It can lead to memory leaks if memory is not deallocated.

B. It can only be used for primitive data types.

C. It can only be used for global variables.

D. It is faster than stack memory allocation.

34.What does the "this" pointer in C++ refer to?

1. It points to the previous object in a linked list.
2. It is a pointer to the current instance of a class.
3. It is used to access static members of a class.
4. It is a reserved keyword for declaring pointers.

35.Which of the following is a common issue related to null pointer assignment in C++?

1. It leads to dangling pointers. b. It leads to dangling pointers.
2. It causes undefined behavior when dereferenced.
3. It increases the program's speed.

36.What is **recursion** in C++?

A. A technique for writing loops B. A way to define infinite functions

C. A function calling itself directly or indirectly

D. A method for passing data between functions

37. In a recursive function, what is the **base case**?

A. The function's return type B. The initial state of the function

C. A condition that stops the recursion D. The last statement in the function

38. What happens if a recursive function lacks a base case?

A. The program will not compile.

B. The function will throw an exception.

C. The recursion will continue indefinitely, leading to a stack overflow.

D. The function will return an error code.

39. What is the term for a recursive function that calls itself **with modified arguments?**

A. Direct recursion B. Tail recursion

C. Indirect recursion D. Recursive function call

40. When should you use recursion over an iterative approach?

A. When you want to simplify the code

B. When you need to improve the program's performance

C. When the problem can be naturally divided into smaller, similar subproblems

D. When you want to avoid using functions

41.In a recursive function, what is the **call stack** used for?

A. To store the return values of function calls

B. To store the local variables of the function

C. To keep track of the function calls and their execution context

D. To optimize the function's performance

42.What is the concept of "**tail recursion**" in C++?

A. It refers to a recursive function that uses a tail to terminate the recursion.

B. It's a way to make recursive functions more readable and efficient by having the recursive call as the last action in the function.

C. It's a technique to prevent recursion in C++.

D. It's an advanced feature only available in C++17 and later.

43.What is the **maximum number of recursive calls** a C++ program can make **before encountering a stack overflow error**?

A. It depends on the amount of available RAM.

B. It depends on the computer's processor speed.

C. It is not limited by the language, but it depends on the system's stack size.

D. It is unlimited; C++ can handle any number of recursive calls.

44.Which type of recursion is typically **less memory-efficient** due to the need to save function call state?

A. Direct recursion B. Tail recursion C. Indirect recursion

D. All types of recursion use the same amount of memory.

45.What is a common **problem that can occur with recursive algorithms?**

A. Excessive memory usage and stack overflow errors

B. Faster execution compared to iterative solutions

C. Inability to handle simple problems

D. Reduced code readability and maintainability.

46.What is the **fundamental concept** of object-oriented programming (OOP)?

A. Variables and data types B. Classes and objects

C. Loops and conditionals D. Functions and procedures

47.In OOP, what is a **class**?

A. A blueprint for creating objects B. A data structure for storing variables

C. A collection of functions D. A program's main function

48.What is an **object** in OOP?

A. A synonym for a class B. An instance of a class

C. A pointer to a function D. A built-in data type

49.What is **encapsulation** in OOP?

A. The process of creating objects from classes

B. The ability to hide the implementation details of an object and restrict access to its data

C. The process of inheritance D. A feature used for code organization

50.In C++, which keyword is used to declare a member of a class that can be accessed without creating an instance of the class?

A. new B. static C. this D. public

51.What is the concept of **message passing** in OOP?

A. A way to send email notifications from objects

B. The process of objects communicating by invoking methods on each other

C. A way to send text messages between objects

D. A feature for displaying messages in a user interface

52.In C++, what is the **purpose of the "private" access modifier** in a class?

A. To make class members accessible from outside the class

B. To restrict access to class members only within the class

C. To make class members accessible from derived classes

D. To create global variables

53.In C++, how do you **specify a class**?

a. Using the class keyword b. Using the struct keyword

c. Using the public keyword d. Using the private keyword

54. What is a **friend function**?

A. A function that is friendly and helpful to other functions

B. A function that can access the private and protected members of a class

C. A function that is declared with the friend keyword

D. A function used for implementing operator overloading

55.What is the **primary purpose of a friend function** in C++?

A. To provide access to a class's public members

B. To provide a way to declare global functions

C. To facilitate function overloading

D. To provide access to a class's private and protected members

56.In C++, where should the **friend function declaration** be placed?

A. In the same class where it is used B. In the derived class

C. In a separate source file D. In the function calling the friend function

57.Can a friend function be a member of the same class it is a friend of?

A. Yes B. No

C. Only if the friend function is static

D. Only if the friend function is declared as a public member

58.What is the **access level of a friend function** in relation to the class it is a friend of?

A. Private B. Protected C. Public D. It has no access level

59.When is it **beneficial to use friend functions** in C++?

A. When you want to hide the implementation details of a class

B. When you need to create a subclass of a class

C. When you want to simplify function overloading

D. When you want to avoid using classes altogether

60.Which **keyword is used to declare a friend function** in a class definition in C++?

A. private B. protected C. friend D. public

61.Can multiple functions be declared as friends of the same class in C++?

A. Yes, but only if they are all declared as static functions.

B. No, a class can have only one friend function.

C. Yes, multiple functions can be declared as friends.

D. No, friend functions cannot be used with class objects.

62.In C++, what is the **arrow operator (->) primarily used for** when working with class objects?

a. Incrementing object values

b. Accessing class methods

c. Performing logical operations

d. Accessing members of an object through a pointer

63.What is a **static member** in a C++ class?

A. A member that can only be accessed within the class

B. A member that belongs to a specific instance of the class

C. A member that is shared among all instances of the class

D. A member that is created dynamically at runtime

64.When you declare a data member as "static" in a class, how many copies of that member are created for all instances of the class?

A. One copy for each instance B. A separate copy for each instance

C. One shared copy for all instances

D. No copy is created; it remains in the class definition only.

65.What is a **static method** in a C++ class?

A. A method that can only be called by objects of the class

B. A method that belongs to a specific instance of the class

C. A method that is shared among all instances of the class

D. A method that cannot be declared in a class

66.What is the **purpose of a static data member** in a C++ class?

A. To make the member accessible only within the class

B. To ensure that each instance of the class has its own copy of the member

C. To allow the member to be accessed without creating an instance of the class

D. To store global variables in the class

67.When is a **static object** constructed in C++?

A. When an instance of the class is created

B. When the program starts

C. When the object is declared with the auto keyword

D. When a dynamic object is deleted

68.What is the **lifespan of a static object** in C++?

A. It has a limited lifespan within a specific scope.

B. It persists throughout the program's execution.

C. It is destroyed when the program terminates.

D. It is recreated each time it is accessed.

69.**Can** a static method access **non-static (regular) members** of a class?

A. Yes, it can access them directly.

B. Yes, but only through an instance of the class.

C. No, it can only access other static members.

D. No, it can't access any members.

70.What is the **significance of a static member or method** in C++?

A. It allows for dynamic memory allocation.

B. It allows for better encapsulation of data.

C. It enables data and functionality that are not tied to a specific instance of a class.

D. It ensures that objects are created with default constructors.