Objectives

○ Develop C++ programs using arrays, structures, and functions with inline and default arguments.
○ Understand and apply pass-by-reference and return-by-reference techniques.
○ Use dynamic memory allocation to manage data at runtime.
○ Solve basic computational problems through structured programming.

Tools and Libraries Used

○ Programming Language: C++
○ IDE: Visual Studio Code
○ Libraries: #include<iostream>, #include<string>

Theory

Arrays:
structure.
When you declare an array like
arrays in C++ are a fundamental way to store multiple elements of the same type in a fixed-size

```
1. int arr[5];
```

you are reserving space for five integers in contiguous memory. These can be initialized directly
using curly braces, for example: int arr[5] = {1, 2, 3, 4, 5}; Each element is accessed using an
index, starting from 0.
Here's a basic example of array usage:

```
1. #include <iostream>
2. using namespace std;
3.
5. 6. 7. 8. }
9. return 0;
10. }
4. int main() {
int numbers[3] = {10, 20, 30};
for (int i = 0; i < 3; ++i) {
cout << "Element " << i << ": " << numbers[i] << endl;
```

Structures:
structures (or struct) in C++ provide a way to group related variables of different types under a
single name. They are useful when modeling real-world entities.
Syntax:

```
1. // Structure definition
2. struct StructName {
3. datatype member1;
4. datatype member2;
5. // ... other members
6. };
7.
8. // Declaring structure variables
9. StructName varName;
10.
11. // Initializing
12. StructName varName = {value1, value2};
13.
14. // Accessing members
15. varName.member1;
16.
```

For instance, if you are dealing with a student record, a structure can be used to combine the
ID, name, and grade into one unit. Here's how a structure is defined and used:

```
1. #include <iostream>
2. using namespace std;
3.
4. struct Student {
5. int id;
6. string name;
7. float grade;
8. };
9.
10. int main() {
11. Student s1 = {1, "Alice", 89.5};
12. cout << "ID: " << s1.id << ", Name: " << s1.name << ", Grade: " << s1.grade << endl;
13. return 0;
14. }
15.
```

In this example, the Student structure contains an integer, a string, and a float. We initialize a
student object and access its members using the dot operator.Functions:
C++ supports both inline functions and functions with default arguments. An inline function is
declared using the inline keyword. It's useful for small, frequently called functions, because the
compiler attempts to expand the function body at the point of call, potentially reducing function-call overhead.
Syntax:

```
1. // Inline function declaration and definition
2. inline returnType functionName(parameterList) {
3. // function body
4. }
```

For example:

```
1. #include <iostream>
2. using namespace std;
3.
4. inline int square(int x) {
5. return x * x;
6. }
7.
8. int main() {
9. cout << "Square of 5: " << square(5) << endl;
10. return 0;
11. }
```

This program outputs the square of 5 using an inline function. Inline functions are best used
when the function body is short and simple.
Default arguments, on the other hand, allow you to call functions without specifying all parameters. Parameters with default values must appear at the end of the parameter list.
Syntax:

```
1. // Function declaration or definition
2. returnType functionName(type param1, type param2 = defaultValue, ...);
3.
4. // Call
5. functionName(value1); // uses default values
6. functionName(value1, value2); // overrides defaults
```

7.For instance:

```
1. #include <iostream>
2. using namespace std;
3.
```

```
5. 6. }
7.
8. int main() {
9. greet("Bob");
10. greet();
11. return 0;
12. }
13.
4. void greet(string name = "Guest") {
cout << "Hello, " << name << "!" << endl;
```

Here, the greet function has a default value of "Guest". If you call greet() without any argument,
it will use the default value.

Parameter passing:

In terms of parameter passing, C++ allows functions to accept parameters by reference. This
means the function can modify the original variable passed to it, since it works directly with the
memory address of the variable. This is achieved using the & symbol in the function's parameter list.

Syntax:

```
1. // Function definition with reference parameter
2. void functionName(datatype ¶m) {
// param can be modified
3. 4. }
```

For example:

```
1. #include <iostream>
2. using namespace std;
3.
4. void increment(int &n) {
5. n++;
6. }
7.
8. int main() {
9. int a = 5;
10. increment(a);
11. cout << "After increment: " << a << endl;
12. return 0;
13. }
```

14.Here, the value of a is modified inside the increment function because it is passed by reference.

In addition to passing by reference, C++ also allows functions to return a reference. This is particularly useful when you want a function to return a modifiable reference to an element in a
data structure like an array.

Syntax:

```
1. // Function returning a reference
2. datatype& functionName(parameters) {
3. // must return a reference to a valid variable
4. }
```

For example:

```
1. #include <iostream>
2. using namespace std;
3.
4. int& getElement(int arr[], int index) {
5. return arr[index];
6. }
7.
8. int main() {
9. int nums[3] = {1, 2, 3};
```

```
10. getElement(nums, 1) = 100;
11. cout << "Updated array: ";
12. for (int i = 0; i < 3; i++) cout << nums[i] << " ";
13. return 0;
14. }
```

This program accesses and modifies an element in the array using a return-by-reference function. The result will show the updated second element as 100.

Dynamic Memory Allocation:

dynamic memory allocation in C++ is used to allocate memory at runtime, which is especially useful when the size of data structures cannot be determined at compile time. You use the new keyword to allocate memory and delete to free it.

Syntax:

```
1. // Single variable
2. datatype* ptr = new datatype;
4. // Array
5. datatype* arr = new datatype[size];
7. // Access
8. *ptr = value;
9. arr[index] = value;
11. // Deallocation
12. delete ptr; 13. delete[] arr; // for single variable
// for arrays
```

For example:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main() {
5. int *ptr = new int;
6. *ptr = 25;
7. cout << "Value: " << *ptr << endl;
8. delete ptr;
9. return 0;
10. }
11.
```

This code dynamically allocates memory for a single integer and stores the value 25 in it.

Similarly, you can create dynamic arrays:

```
1. #include <iostream>
2. using namespace std;
3.
4. int main() {
5. int size = 4;
6. int *arr = new int[size];
7. for (int i = 0; i < size; i++) {
8. arr[i] = i + 1;
9. }
10. 11. for (int i = 0; i < size; i++) {
cout << arr[i] << " ";
12. }
13. delete[] arr;
14. return 0;
15. }
16.
```

The program allocates space for an array of integers, initializes them, prints them, and then deallocates the memory. This is critical for memory management in larger programs.

Q1: Write a C++ program to overload a function add() to handle:
Two integers, Two floats, One integer and one float

Code:

```
1. #include <iostream>
2. using namespace std;
3. // Function to add two integers
4. int add(int a, int b) {
5. return a + b;
6. }
7. // Function to add two floats
8. float add(float x, float y) {
9. return x + y;
10. }
11. // Function to add one integer and one float
12. float add(int a, float b) {
13. return a + b;
14.
15. }
16. int main() {
17. int a = 5, b = 10;
18. float x = 3.5f, y = 2.5f;
19.
20. cout << "Sum of two integers: " << add(a, b) << endl;
21. cout << "Sum of two floats: " << add(x, y) << endl;
22. cout << "Sum of one int and one float: " << add(a, y)
23. << endl;
24.
25. return 0;
26. }
27.
```

Output:

```
1. Sum of two integers: 15
2. Sum of two floats: 6
3. Sum of one int and one float: 7.5
```

Q2: Write an inline function in C++ to calculate the square of a number and demonstrate it with at least two function calls.

Code:

```
1. //inline functions
2. #include <iostream>
3. using namespace std;
4.
5. inline int square(int a){
6. return a*a;
7. }
8.
9. int main(){
10. int n1, n2;
11.
12. cout << "Enter two number: ";
13. cin >> n1 >> n2;
14.
15. cout << "Square of " << n1 << " is " << square(n1) << endl;
16. cout << "Square of " << n2 << " is " << square(n2) << endl;
17. return 0;
18. }
19.
```

Output:

1. Enter two number: 5
2. 8
3. Square of 5 is 25
4. Square of 8 is 64
5.

Q3: Write a program using a function with default arguments for calculating total price. The function should take the item price and quantity, with quantity defaulting to 1.

Code:

```
1. #include<iostream>
2. using namespace std;
3. // default argument
4.
6. 5. void output(int quantity = 1, double price = 0.0){
cout << "Total: " << quantity * price << endl;
7. }
8. int main(){
9.
10. int quantity;
11. double price;
12. cout << "Enter quantity: ";
13. cin >> quantity;
14. cout << "Enter price: ";
15. cin >> price;
16. output(quantity, price);
17. return 0;
18. }
19.
```

Output:

1. Enter quantity: 2
2. Enter price: 4
3. Total: 8
4

.Q4: Write a C++ program to swap two numbers using pass-by-reference.

Code:

```
1. // pass by reference
2. #include<iostream>
3. using namespace std;
4.
5. void swap(int &x, int &y);
6.
7. int main(){
8. int a = 1, b = 2;
9. cout << "Before swap: a = " << a << ", b = " << b << endl;
10. swap(a, b);
11. cout << "After swap: a = " << a << ", b = " << b << endl;
12.
13. return 0;
14. }
15.
16. void swap(int &x, int &y){
17. int temp = x;
18. x = y;
19. y = temp;
20. }
21.
```

Output:

```
1. Before swap: a = 1, b = 2
2. After swap: a = 2, b = 1
3.
```

Q5: Create a function that returns a reference to an element in an array and modifies it.

Code:

```
1. #include <iostream>
2. using namespace std;
3. // Function returns reference to an array element
4. int& getElement(int arr[], int index) {
5. return arr[index]; // returning reference
6. }
7. int main() {
8. int numbers[5] = {10, 20, 30, 40, 50};
9. cout << "Original value at index 2: " << numbers[2] << endl;
10. // Modify the element using returned reference
11. getElement(numbers, 2) = 99;
12. cout << "Modified value at index 2: " << numbers[2] << endl;
13. return 0;
14. }
15.
```

Output:

```
1. Modified value at index 2: 99
2.
```

Q6: Write a program to input 5 integers in an array and print their squares using a pointer.

Code:

```
1. #include<iostream>
2. using namespace std;
3. int main(){
4. int arr[5];
5. int* ptr = arr;
6. cout << "enter 5 integers:\n";
7. for(int i=0;i<5;i++){
8. cin >> *(ptr+i);
9. }
10.
11. cout << "The square of the integers are:\n";
12. for(int i=0; i<5; i++) {
13. cout << *(ptr+i) << " ^ 2 = " << ((*(ptr+i)) * (*(ptr+i))) << endl;
14. }
15. return 0;
16. }
```

Output:

```
1. enter 5 integers:
2. 2
3. 3
4. 6
5. 8
6. 9
7. The square of the integers are:
8. 2 ^ 2 = 4
9. 3 ^ 2 = 9
10. 6 ^ 2 = 36
11. 8 ^ 2 = 64
12. 9 ^ 2 = 81
```

Q7: Define a structure Student with data members roll, name, and marks. Input and display
details of 3 students.

Code:

```
#include <iostream>
using namespace std;
struct Student {
int roll;
string name;
float marks;
};
int main() {
Student s[3]; // Array of 3 students
// Input details
for (int i = 0; i < 3; i++) {
cout << "\nEnter details for student " << i + 1 << ":\n";
cout << "Roll number: ";
cin >> s[i].roll;
cin.ignore(); // Clear input buffer
cout << "Name: ";

getline(cin, s[i].name);
cout << "Marks: ";
cin >> s[i].marks;
}
// Display details
cout << "\nStudent Details:\n";
for (int i = 0; i < 3; i++) {
cout << "Student " << i + 1 << ": ";
cout << "Roll = " << s[i].roll << ", Name = " << s[i].name <<
", Marks = " << s[i].marks << endl;

}

return 0;
}
```

Output:

```
Enter details for student 1:
Roll number: 22
Name: Glaive
Marks: 69

Enter details for student 2:
Roll number: 23
Name: sova
Marks: 78

Enter details for student 3:
Roll number: 6
Name: ram
Marks: 57

Student Details:
Student 1: Roll = 22, Name = Glaive, Marks = 69
Student 2: Roll = 23, Name = sova, Marks = 78
Student 3: Roll = 6, Name = ram, Marks = 57
```

Q8: Write a C++ program to demonstrate the difference between structure and union by declaring the same data members and showing memory usage.

Code:

```
1. #include <iostream>
2. using namespace std;
3. struct StudentStruct {
4. int roll;
5. char grade;
6. float marks;
7. };
8. union StudentUnion {
9. int roll;
10. char grade;
11. float marks;
12. };
13. int main() {
14. StudentStruct s;
15. StudentUnion u;
16. cout << "Size of Structure: " << sizeof(s) << " bytes" << endl;
17. cout << "Size of Union : " << sizeof(u) << " bytes" << endl;
18.
19. // Assigning values
20. s.roll = 101;
21. s.grade = 'A';
22. s.marks = 89.5;
23. cout << "\nStructure Data:\n";
24. cout << "Roll: " << s.roll << ", Grade: " << s.grade << ", Marks: " << s.marks <<
endl;
25. // Assigning values to union (only last assigned will retain correct value)
26. u.roll = 101;
27. u.grade = 'A';
28. u.marks = 89.5;
29. cout << "\nUnion Data (only last assigned value is reliable):\n";
30. cout << "Roll: " << u.roll << " (corrupted), Grade: " << u.grade << " (corrupted),
Marks: " << u.marks << endl;
31. return 0;
32. }
33.
```

Output:

```
1. Size of Structure: 12 bytes
2. Size of Union : 4 bytes
3.
4. Structure Data:
5. Roll: 101, Grade: A, Marks: 89.5
6.
7. Union Data (only last assigned value is reliable):
8. Roll: 1119027200 (corrupted), Grade: (corrupted), Marks: 89.5
9.
```

Q9: Create an enum for days of the week. Display a message depending on the selected day.

Code:

```
1. #include<iostream>
2. using namespace std;
3.
4. enum Day {sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
5.
6. int main(){
7. Day today;
8. int choice;
9. cout << "Enter a number (0-6) for the day of the week: ";
10. cin >> choice;
11. today = static_cast<Day>(choice);
```

```
12. switch(today){
13. case sunday:
14. cout << "Today is Sunday" << endl;
15. break;
16. case Monday:
17. cout << "Today is Monday" << endl;
18. break;
19. case Tuesday:
20. cout << "Today is Tuesday" << endl;
21. break;
22. case Wednesday:
23. cout << "Today is Wednesday" << endl;
24. break;
25. case Thursday:
26. cout << "Today is Thursday" << endl;
27. break;
28. case Friday:
29. cout << "Today is Friday" << endl;
30. break;
31. case Saturday:
32. cout << "Today is Saturday" << endl;
33. break;
34. default:
35. cout << "Invalid choice" << endl;
36. break;
37. }
38. return 0;
39. }
```

Output:

```
1. Enter a number (0-6) for the day of the week: 5
2. Today is Friday
```

Q10: Write a C++ program to allocate memory for an array of integers using new, input values,
calculate their sum, and free the memory using delete.

Code:

```
1. #include<iostream>
2. using namespace std;
3.
4. int main() {
5. int n;
6. cout << "enter number of elements: ";
7. cin >> n;
8.
9.
10. int* arr = new int[n];
11. cout << "enter " << n << " integers:\n";
12. for(int i=0; i<n; i++) {
13. cin >> arr[i];
14. }
15.
16. int sum=0;
17. for(int i = 0; i < n; i++)
18. {
19. sum += arr[i];
20. }
21.
22. cout << "sum of the array elements ="<<sum<<endl;
23.
24. delete[] arr;
25.
26. return 0;
```

```
27. }
```

Output:

```
1. enter number of elements: 4
2. enter 4 integers:
3. 32
4. 34
5. 2
6. 1
7. sum of the array elements =69
```

Conclusion:

Together, these concepts—arrays, structures, functions (with inline and default arguments),reference passing and returning, and dynamic memory allocation—form a strong foundation forsolving a wide range of computational problems in C++. They are not just theoretical featuresbut practical tools that improve code organization, efficiency, and scalability.