Objective:

- Understand about array, function, inline function, default argument, structure, pass by reference, pass by value, Dynamic Memory Allocation
- Solve problem using them.

Tools and Libraries Used:

- Programming Language: C++
- IDE: Any C++ compiler (G++, Code::Blocks, etc.)
- Libraries: #include <iostream>, #include <cmath>, #include <string>

Theory:

Array

An array is a collection of fixed-size elements of the same data type stored in contiguous memory locations. It allows easy access using indices.

Syntax:

```
int arr[5] = {1, 2, 3, 4, 5};

cout << arr[2]; // Outputs 3
```

Function

A function is a block of code that performs a specific task. It promotes code reuse and better structure.

Syntax:

```
int add(int a, int b) {
```

```
    return a + b;

}
```

Inline_Function

An inline function tells the compiler to insert the complete function code wherever it is called, reducing function call overhead (used for small functions).

Syntax:

```
inline int square(int x) {

    return x * x;

}
```

Default Argument

Default arguments are values provided in the function declaration. If the caller omits them, the defaults are used.

Syntax:

```
int add(int a, int b = 5) {

    return a + b;

}
```

Structure

A structure (struct) is a user-defined data type that groups related variables of different types.

Syntax:

```
struct Student {
```

```
    int id;
    string name;
};
```

Pass by Reference:

Passing by reference allows the function to modify the original variable using references (alias to original).

Syntax:

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

Return_by_Reference

A function can return a reference so the returned value can be modified directly.

Syntax:

```
int& getElement(int arr[], int index) {
    return arr[index];
}
```

Dynamic Memory Allocation

Dynamic memory is allocated during runtime using new and released using delete.

Syntax:

int* p = new int; // allocate

*p = 10;

delete p; // deallocate

Qn1. Write C++ program to overload a function add() to handle two integer , Two float , one integer and one float.

Code:

```
#include<iostream>
using namespace std;
int add(int a,int b){
return a+b;
}
float add(float x, float y){
return x+y;}
float add(int a, float x){
return a+x;
}

int  main(){
int a=6,b=24;
float x=2.5f,y=5.5f;
cout<<"Sum of two integer:"<<add(a,b)<<endl;
cout<<"Sum of two floats:"<<add(x,y)<<endl;
cout<<"Sum of one integer and one float:"<<add(a,x)<<endl;
return 0;
}
```

Output:

```
Sum of two integer:30
Sum of two floats:8
Sum of one integer and one float:8.5
```

Qn2 : Write an inline function in C++ to calculate the square of a number and demonstrate it with at least two function call.

Code:

```
#include<iostream>
using namespace std;
inline int square(int a){
return a*a;}
int main(){
    int n1,n2;
cout<<"Enter number for first function call:";
cin>>n1;
cout<<"Square of "<<n1<<" is:"<<square(n1)<<endl;
cout<<"Enter number for second function call:";
cin>>n2;
cout<<"Square of "<<n2<<" is:"<<square(n2)<<endl;
return 0;
}
```

Output:

```
Enter number for first function call:5
Square of 5 is:25
Enter number for second function call:14
Square of 14 is:196
```

Qn 3: Write a program using function with default argument for calculating total price. The function should take item price and quantity, with quantity defaulting to 1.

Code:

```cpp
#include<iostream>
using namespace std;
float calcTotal(float price, int quantity=1){
return price*quantity;
}
int main(){
float itmprice=1499.55;
cout<<"Total price for 1 item:"<<calcTotal(itmprice)<<endl;
cout<<"Total price for 7 item:"<<calcTotal(itmprice,7)<<endl;
return 0;
}
```

Output:

```
Total price for 1 item:1499.55
Total price for 7 item:10496.9
```

Qn 4. Write a C++ program to swap two number using pass-by-reference.

Code:

```cpp
#include<iostream>
using namespace std;
void swapnum(int &a, int &b){
int temp=a;
a=b;
b=temp;
}
int main(){
int a,b;
cout<<"Enter 1st Number:";
cin>>a;
cout<<"Enter 2nd Number:";
cin>>b;
cout<<"Before swap a="<<a<<",b="<<b<<endl;
swapnum(a,b);
cout<<"After swap a="<<a<<",b="<<b<<endl;
return 0;
}
```

Output:

```
Enter 1st Number:6
Enter 2nd Number:7
Before swap a=6,b=7
After swap a=7,b=6
```

Qn 5. Create a function that returns a reference to an element in an array and modifies it.

Code:

```
#include<iostream>
using namespace std;
int& getelem(int arr[],int index){
return arr[index];
}
int main(){
int num[6]={5,6,27,56,2};
cout<<"Original Value at index 3:"<<num[3]<<endl;
getelem(num,3)=93;
cout<<"Modified value at index 3:"<<num[3]<<endl;
return 0;
}
```

Output:

```
Original Value at index 3:56
Modified value at index 3:93

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Qn 6. Write a program to input 5 integer in an array and print their squares using pointer.

Code:

```
#include<iostream>
using namespace std;
int main(){
int arr[5];
int *ptr=arr;
cout<<"Enter 5 integers:";
for(int i=0;i<5;i++){
    cin>>*(ptr+i);
}
cout<<"\nSquare of the entered element are:"<<endl;
for(int i=0;i<5;i++){
    cout<<*(ptr+i)<<"^2="<<*(ptr+i)**(ptr+i)<<endl;
}
return 0;
}
```

Output:

```
Enter 5 integers:1
2
3
4
5

Square of the entered element are:
1^2=1
2^2=4
3^2=9
4^2=16
5^2=25
```

Qn 7. Define a structure Student with data member roll, name, and marks. Input and display details of 3 Student.

Code:

```cpp
#include<iostream>
using namespace std;
struct student{
int roll;
string name;
float marks;
};
int main(){
struct student s[3];
for(int i=0;i<3;i++){
    cout<<"\nEnter the details of student "<<i+1<<endl;
    cout<<"Roll No:";
    cin>>s[i].roll;
    cin.ignore();
    cout<<"Name:";
    getline(cin,s[i].name);
    cout<<"Marks:";
    cin>>s[i].marks;
}
cout<<"Student Details:"<<endl;
for(int i=0;i<3;i++){
    cout<<"Student "<<i+1<<endl;
    cout<<"Roll="<<s[i].roll<<"Name="<<s[i].name<<"Marks="<<s[i].marks<<endl;
}
return 0;
```

Output:

```
Enter the details of student 1
Roll No:4
Name:afraed
Marks:45

Enter the details of student 2
Roll No:3
Name:gdfsgds
Marks:54

Enter the details of student 3
Roll No:1
Name:sgdsfdfh
Marks:78
Student Details:
Student 1
Roll=4Name=afraedMarks=45
Student 2
Roll=3Name=gdfsgdsMarks=54
Student 3
Roll=1Name=sgdsfdfhMarks=78
```

Qn 8. Write a C++ program to demonstrate the difference between structure and union by declaring the same data members showing memory usages.

Code:

```
using namespace std;
struct studentStruct{
int roll;
char grade;
float marks;
};
union studentUnion{
int roll;
char grade;
float marks;
};
int main(){
struct studentStruct s;
union studentUnion u;
cout<<"Size of Structure:"<<sizeof(s)<<"bytes"<<endl;
cout<<"Size of Union:"<<sizeof(u)<<"bytes"<<endl;
s.roll=9;
s.grade='A';
s.marks=96.7;
cout<<"\nStudent Details:"<<endl;
cout<<"Roll="<<s.roll<<" Grade="<<s.grade<<", Marks="<<s.marks<<endl;
u.roll=9;
u.grade='A';
u.marks=96.7;
cout<<"\nUnion Details:"<<endl;
cout<<"Roll="<<u.roll<<"(corrupted),Grade="<<u.grade<<" (corrupted),Marks="<<u.marks<<endl;


return 0;
}
```

Output:

```
Size of Structure:12bytes
Size of Union:4bytes

Student Details:
Roll=9 Grade=A, Marks=96.7

Union Details:
Roll=1119970918(corrupted),Grade=f (corrupted),Marks=96.7
```

Qn 9. Create an enum for days of week. Display a message depending upon the selected day.

Code:

```
#include <iostream>
using namespace std;
enum Day {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
int main() {
    int choice;
    cout << "Enter day number (0 = Sunday, 1 = Monday, ..., 6 = Saturday): ";
    cin >> choice;
    if (choice < 0 || choice > 6) {
        cout << "Invalid day number!" << endl;
        return 0;
    }
    Day today = (Day)choice;
    switch (today) {
        case Sunday:
            cout << "Relax! It's Sunday." << endl;
            break;
        case Monday:
            cout << "Back to work! It's Monday." << endl;
            break;
        case Tuesday:
            cout << "Keep going! It's Tuesday." << endl;
            break;
        case Wednesday:
            cout << "Midweek! It's Wednesday." << endl;
            break;
        case Thursday:
            cout << "Almost Friday! It's Thursday." << endl;
            break;
        case Friday:
            cout << "Weekend is near! It's Friday." << endl;
            break;
        case Saturday:
            cout << "Enjoy! It's Saturday." << endl;
            break;
    }
    return 0;
}
```

Output:

```
Enter day number (0 = Sunday, 1 = Monday, ..., 6 = Saturday): 3
Midweek! It's Wednesday.

Process returned 0 (0x0)   execution time : 6.061 s
Press any key to continue.
```

Qn 10. Write a C++ program to allocate memory for an array of integer using new, input values, calculate their sum, and free the memory usage.

Code:

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

int main() {
    int size, sum = 0;

    cout << "Enter the number of elements: ";
    cin >> size;

    // Allocate memory for array using new
    int* arr = new int[size];

    // Input values
    cout << "Enter " << size << " numbers:" << endl;
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
        sum += arr[i];
    }

    // Display sum
    cout << "Sum of the numbers is: " << sum << endl;

    // Free the memory
    delete[] arr;

    return 0;
}
```

Output:

```
Enter the number of elements: 5
Enter 5 numbers:
1
2
3
4
5
Sum of the numbers is: 15
```

Discussion:

In this C++ lab, we practiced using arrays to store data, and functions to organize code. We implemented inline functions for faster execution and used default arguments for flexible function calls. Structures helped us group related variables. We also learned pass by reference to modify values directly and return by reference to return actual variables. Lastly, we used dynamic memory allocation with new and delete for efficient memory usage in arrays.

Conclusion:

In conclusion, this lab helped us understand and apply key C++ concepts that are essential for effective programming. We learned how arrays can store multiple values, and how functions improve code modularity. Inline functions and default arguments made our programs more efficient and flexible. Structures allowed us to manage grouped data types easily. We also understood the power of pass by reference and return by reference for memory-efficient coding. Lastly, dynamic memory allocation using new and delete gave us control over memory, making our programs more efficient and suitable for real-time applications.