Course Code: - CSM-6252
Course Name: - DAA and Web Programming Lab

MASTER OF COMPUTER APPLICATIONS (MCA)

PROGRAMME DESIGN COMMITTEE

Prof. Masood Parveez
Vice Chancellor – Chairman
MTSOU, Tripura

Prof. Abdul Wadood Siddiqui
Dean Academics
MTSOU, Tripura

Prof. C.R.K. Murty
Professor of Distance Education
IGNOU, New Delhi

Prof. Mohd. Nafees Ahmad Ansari
Director of Distance Education
Aligarh Muslim University, Aligarh

Prof. P.V. Suresh
Professor of Computer Science
IGNOU, New Delhi

Prof. V.V. Subrahmanyam
Professor of Computer Science
IGNOU, New Delhi

Prof. S. Nagakishore Bhavanam
Professor of Computer Science
Mangalayatan University, Jabalpur

Prof. Manoj Varshney
Professor of Computer Science
MTSOU, Tripura

COURSE WRITERS

Dr. Md. Amir Khusru Akhtar
Associate Professor of Computer Science
MTSOU, Tripura
CSM-6211 Web Programming

Dr. Ankur Kumar
Assistant Professor
MTSOU, Tripura
CSM-6212 Advance Cyber Security

Dr. Manish Saxena
Assistant Professor of Computer Science
MTSOU, Tripura
CSM-6213 Management Information & system

Dr. Duvvuri B. K. Kamesh
Assistant Professor of Computer Science
MTSOU, Tripura
CSM-6214 Design & Analysis of Algorithm

Mr. Pankaj Kumar
Assistant Professor of Computer Science
Mangalayatan University, Aligarh
CSM-6251 Data Structure using C++ & Lab

Dr. Manoj Varshney
Associate Professor of Computer Science
MTSOU, Tripura
CSM-6252 DAA and Web Programming Lab

COURSE EDITORS

Prof. S. Nagakishore Bhavanam
Professor of Computer Science
Mangalayatan University, Jabalpur

Prof. Jawed Wasim
Associate Professor of Computer Science
Mangalayatan University, Aligarh

Dr. Manoj Varshney
Associate Professor of Computer Science
MTSOU, Tripura

Dr. M. P. Mishra
Associate Professor of Computer Science
IGNOU, New Delhi

Dr. Akshay Kumar
Associate Professor of Computer Science
IGNOU, New Delhi

FORMAT EDITORS

Dr. Nitendra Singh
Associate Professor of English
MTSOU, Tripura

Ms. Angela Fatima Mirza
Assistant Professor of English
MTSOU, Tripura

Dr. Faizan
Assistant Professor of English
MTSOU, Tripura

Ms. Vanshika Singh
Assistant Professor of English
MTSOU, Tripura

MATERIAL PRODUCTION

1. Mr. Himanshu Saxena
2. Ms. Rainu Verma
3. Mr. Jeetendra Kumar
4. Mr. Khiresh Sharma
5. Mr. Ankur Kumar Sharma
6. Mr. Pankaj Kumar

CONTENT

Page No.

CSM–6252: DATA STRUCTURE USING C++ LAB

Structure

1.0 INTRODUCTION

C++ is a powerful, high-performance programming language widely used in software development, system programming, game development, and real-time simulations. Its efficiency and control over system resources make it an essential tool for professional programmers. By learning C++, you will gain a strong foundation in programming concepts that apply to many other languages and development environments.

Data structures are critical components in computer science and software engineering, as they enable efficient storage, retrieval, and modification of data. Understanding how to implement and utilize data structures like arrays, linked lists, stacks, queues, trees, and graphs is fundamental to developing robust and efficient software solutions.

1.1 OBJECTIVES

The lab sessions in this manual are structured to achieve the following objectives:

1. Reinforce Theoretical Concepts: Apply theoretical knowledge from lectures in a practical, hands-on environment.

2. Develop Problem-Solving Skills: Enhance your ability to solve complex problems by breaking them down into manageable tasks.

3. Understand Implementation Details: Gain a deeper understanding of how data structures are implemented and optimized in C++.

4. Improve Programming Proficiency: Increase your proficiency in C++ through practice and real-world application.

1.2 LAB SETUP REQUIREMENT

Software Requirements:

- Compiler: GCC (g++), Clang, or any C++ compiler
- IDE: Visual Studio Code, Code: Blocks, CLion, or any C++ IDE
- OS: Windows, Linux, or macOS

Steps to Set Up:

❖ Install a C++ compiler.

Detailed Explanation

Here is the step-by-step process to download and install Dev C++ Compiler

Step 1: Open google.com in the browser. Search for Dev C++ download as shown in the image below.

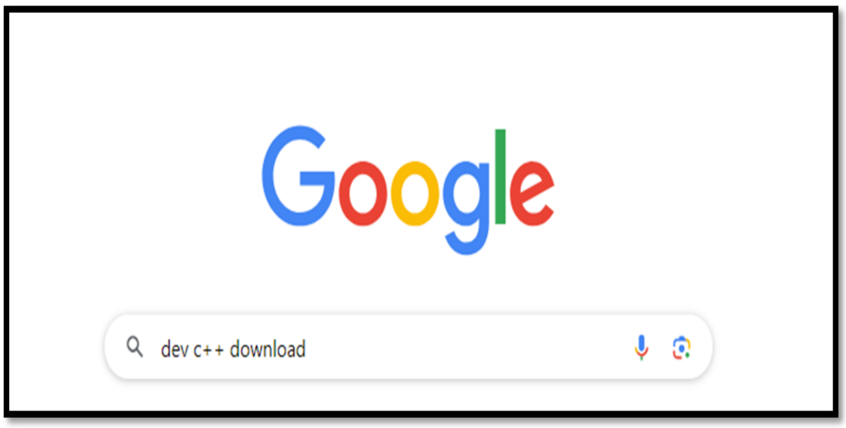
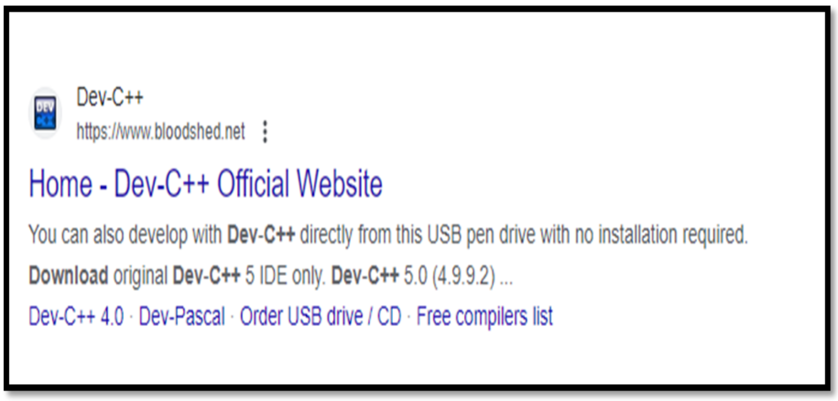


Figure 1: Google Search

Step 2: Click on the link as shown below.



Dev-C++
https://www.bloodshed.net

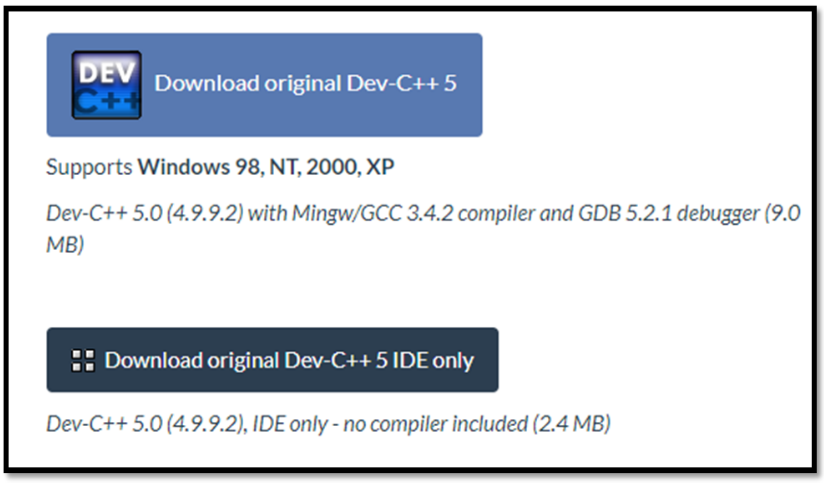
Home - Dev-C++ Official Website

You can also develop with **Dev-C++** directly from this USB pen drive with no installation required. **Download** original **Dev-C++** 5 IDE only. **Dev-C++** 5.0 (4.9.9.2) ...

Dev-C++ 4.0 · Dev-Pascal · Order USB drive / CD · Free compilers list

Figure 2: Search Result

Step 3: Click on the Download button as shown in the image below.



Download original Dev-C++ 5

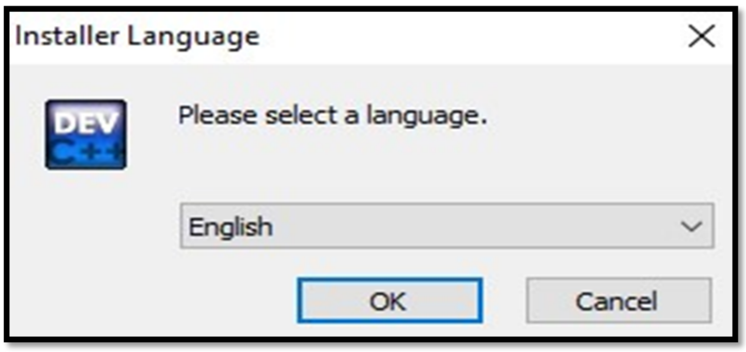
Supports **Windows 98, NT, 2000, XP**

Dev-C++ 5.0 (4.9.9.2) with Mingw/GCC 3.4.2 compiler and GDB 5.2.1 debugger (9.0 MB)

Download original Dev-C++ 5 IDE only

Dev-C++ 5.0 (4.9.9.2), IDE only - no compiler included (2.4 MB)

Step 4: Now double-click on the downloaded file and proceed with the installation as shown in the images below.



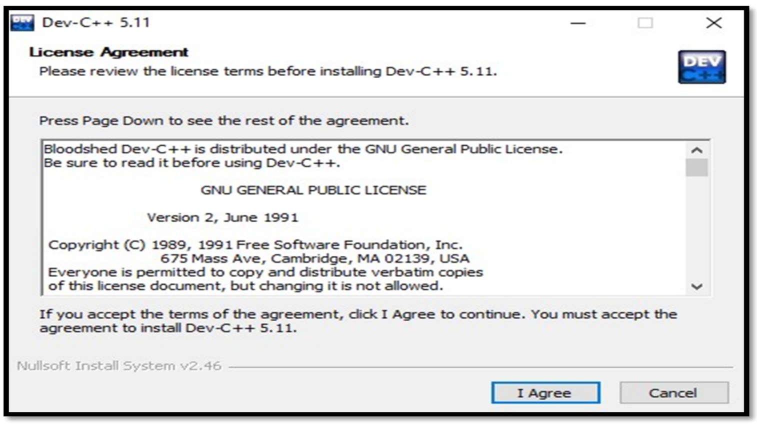
Installer Language

Please select a language.

English

Figure 3: Click on OK.

Step 5: Click on I Agree.



License Agreement

Please review the license terms before installing Dev-C++ 5.11.

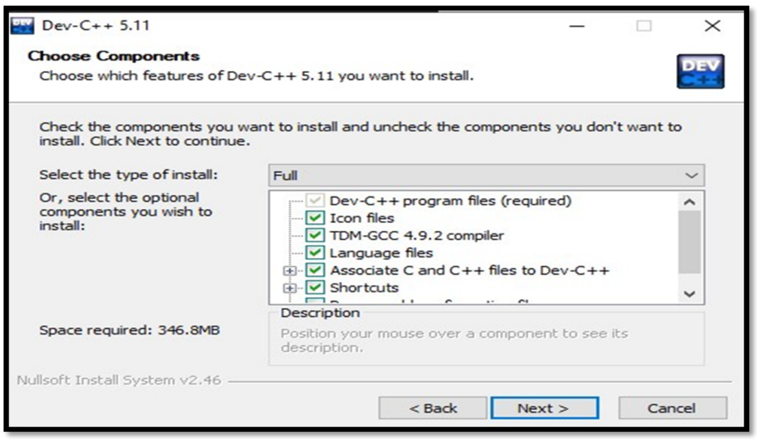
Press Page Down to see the rest of the agreement.

```
Bloodshed Dev-C++ is distributed under the GNU General Public License.
Be sure to read it before using Dev-C++.

                    GNU GENERAL PUBLIC LICENSE

              Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
                675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.
```

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Dev-C++ 5.11.

Nullsoft Install System v2.46

Step 6: Click on Next.



Choose Components

Choose which features of Dev-C++ 5.11 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Full

Or, select the optional components you wish to install:

- ☑ Dev-C++ program files (required)
- ☑ Icon files
- ☑ TDM-GCC 4.9.2 compiler
- ☑ Language files
- ☑ Associate C and C++ files to Dev-C++
- ☑ Shortcuts

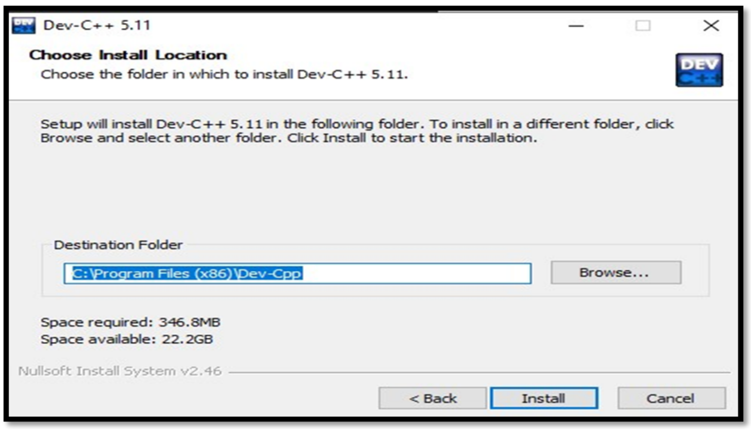
Description

Position your mouse over a component to see its description.

Space required: 346.8MB

Nullsoft Install System v2.46

Step 7: Click on Install. The image below is how it looks during installation.



Choose Install Location

Choose the folder in which to install Dev-C++ 5.11.

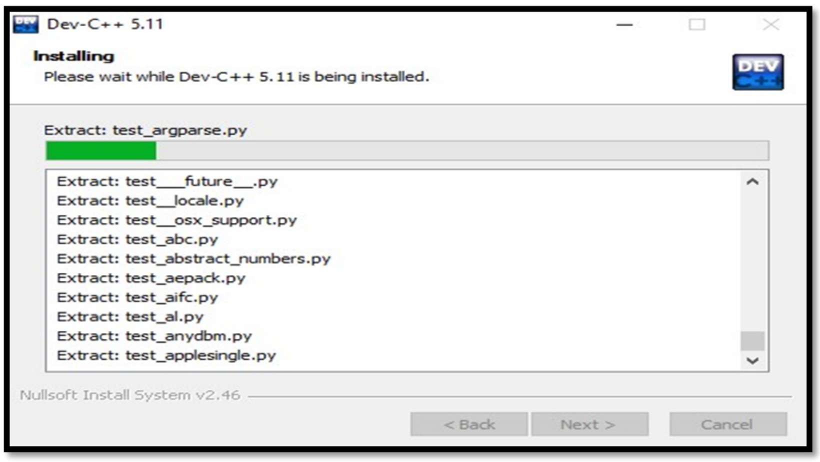
Setup will install Dev-C++ 5.11 in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Dev-Cpp

Space required: 346.8MB

Space available: 22.2GB

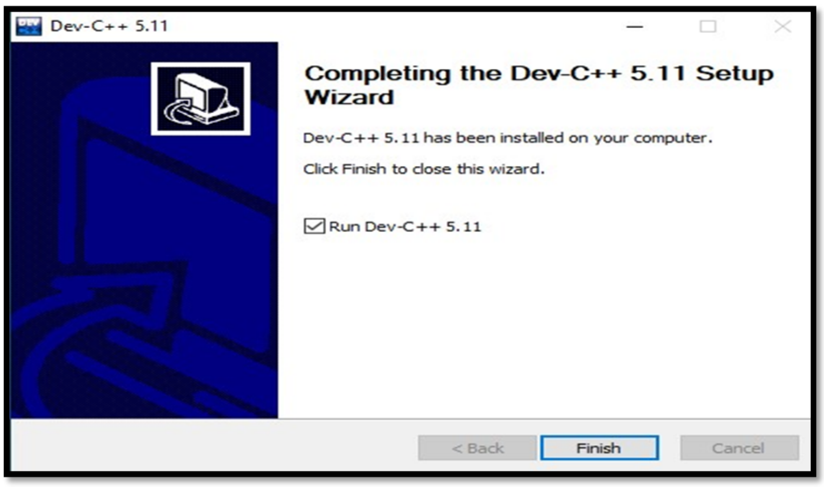
Nullsoft Install System v2.46


Dev-C++ 5.11

Installing
Please wait while Dev-C++ 5.11 is being installed.

```
Extract: test_argparse.py

Extract: test___future__.py
Extract: test__locale.py
Extract: test__osx_support.py
Extract: test_abc.py
Extract: test_abstract_numbers.py
Extract: test_aepack.py
Extract: test_aifc.py
Extract: test_al.py
Extract: test_anydbm.py
Extract: test_applesingle.py
```

Step 8: Click on Finish.


Dev-C++ 5.11

Completing the Dev-C++ 5.11 Setup Wizard

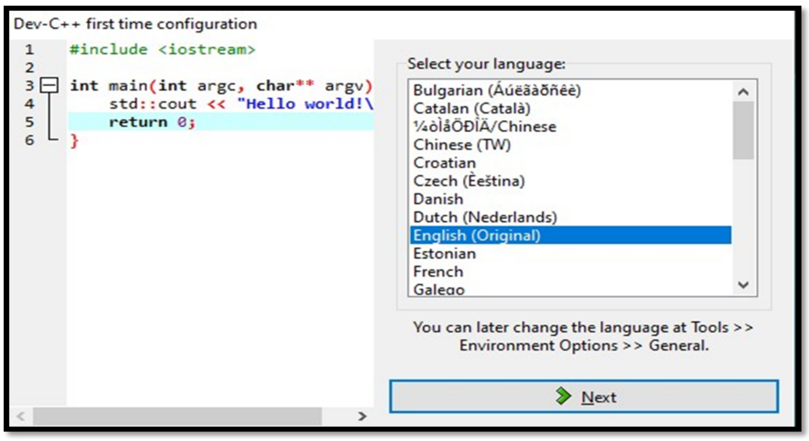
Dev-C++ 5.11 has been installed on your computer.

Click Finish to close this wizard.

☑ Run Dev-C++ 5.11

After clicking on the finish, we will be prompted with a configuration wizard as shown below. Now click on next.

Step 10: Click on Next


Dev-C++ first time configuration

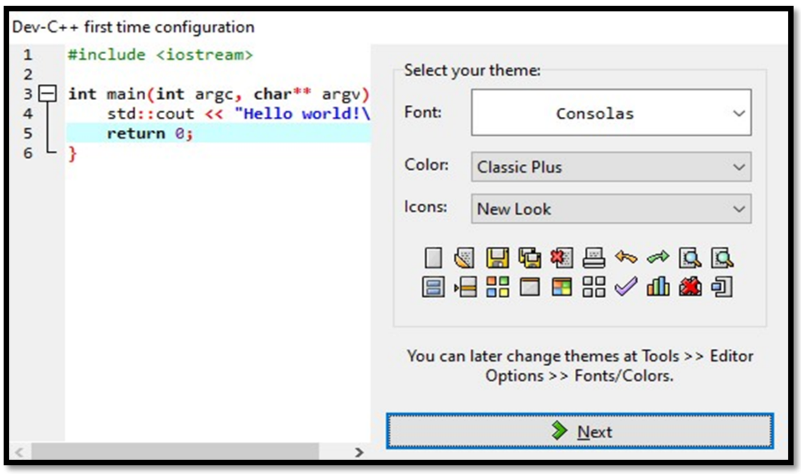
```
#include <iostream>

int main(int argc, char** argv)
    std::cout << "Hello world!\
    return 0;
}
```

Select your language:

- Bulgarian (Áúëãàðñêè)
- Catalan (Català)
- ¼òÌåÖÐÎÄ/Chinese
- Chinese (TW)
- Croatian
- Czech (Èeština)
- Danish
- Dutch (Nederlands)
- English (Original)
- Estonian
- French
- Galego

You can later change the language at Tools >> Environment Options >> General.

Step 11: Click on Next again.



Dev-C++ first time configuration

```
#include <iostream>

int main(int argc, char** argv)
    std::cout << "Hello world!\
    return 0;
}
```

Select your theme:

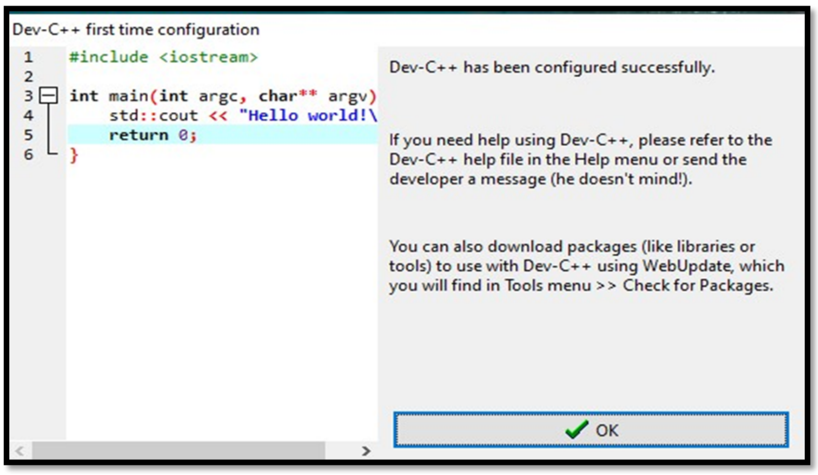
Font: Consolas

Color: Classic Plus

Icons: New Look

You can later change themes at Tools >> Editor Options >> Fonts/Colors.

Step 12: Click on OK.



Dev-C++ first time configuration

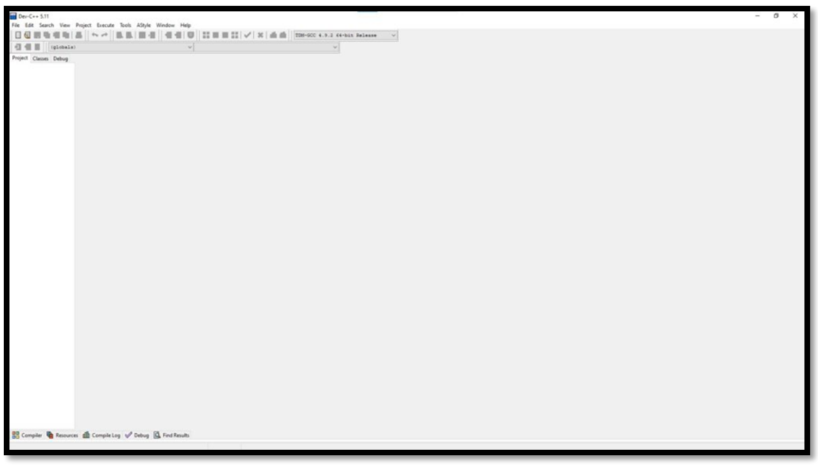
```
#include <iostream>

int main(int argc, char** argv)
    std::cout << "Hello world!\
    return 0;
}
```

Dev-C++ has been configured successfully.

If you need help using Dev-C++, please refer to the Dev-C++ help file in the Help menu or send the developer a message (he doesn't mind!).

You can also download packages (like libraries or tools) to use with Dev-C++ using WebUpdate, which you will find in Tools menu >> Check for Packages.

Now the installation is completed, and the editor will be opened as shown in the image below.



Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
Compiler Resources Compile Log Debug Find Results

- **Install your preferred IDE.**

  List of IDES –

  - Visual Studio Code
  - Code: Blocks
  - CLion
  - Eclipse
  - Code Lit

- **Configure your IDE to recognize the compiler.**

- **Create a workspace for your lab exercises.**

1.3 LAB SESSION OVERVIEW

The lab sessions in this course offer a comprehensive exploration of C++ programming and data structures. Each session is carefully structured to build upon the previous one, progressively advancing from fundamental concepts to more complex topics. Students will engage in hands-on exercises designed to reinforce theoretical knowledge, develop problem-solving skills, and enhance programming proficiency. The sessions cover a wide range of topics, including basic syntax and control structures in C++, functions, recursion, arrays, strings, pointers, object-oriented programming principles such as classes, inheritance, and polymorphism, as well as fundamental data structures like linked lists, stacks, queues, trees, and graphs. Through a combination of theoretical explanations, practical exercises, and guided experimentation, students will gain a deep understanding of both C++ programming and the implementation and application of various data structures. The lab sessions are supplemented with a structured lab report template, providing students with an opportunity to document their work, results, and reflections, and facilitating assessment based on their understanding and proficiency demonstrated throughout the course. Overall, the lab sessions aim to equip students with essential programming skills and knowledge that will serve as a solid foundation for their future studies and careers in computer science and software engineering.

1.4 LAB EXPERIMENTS

1.4.1 Lab 1: Introduction to C++

Objective: Understand the basic structure of a C++ program, compilation, and execution.

1. Write a simple C++ program to display "Hello, World!".

Code:

```
#include <iostream>

using namespace std;

int main () {

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

    return 0;

}
```

Practice Question: Understand the usage of **#include**, **main ()**, and basic I/O (**cin** and **cout**).

1.4.2 Lab 2: Control Structures

Objective: Learn the use of conditional statements and loops.

2. Write a program to check if a number is even or odd.

Code Example:

```
#include <iostream>

using namespace std;
```

```
int main () {
    int num;
    cout << "Enter an integer: ";
    cin >> num;
    if (num % 2 == 0)
        cout << num << " is even." << endl;
    else
        cout << num << " is odd." << endl;
    cout << endl;
    return 0;
}
```

Practice Question: Write a program to print the first 10 natural numbers using a **for** loop.

Practice Question: Write a program to print this pattern**-**

```
            *
        *   *   *
    *   *   *   *   *
*   *   *   *   *   *   *
```

1.4.3 Lab 3: Functions and Recursion

Objective: Understand functions, parameter passing, and recursion. **Tasks**:

1. Write a function to calculate the factorial of a number.

Code:

```
#include <iostream>

using namespace std;

int factorial (int n) {

   if (n == 0)

      return 1;

   else

      return n * factorial (n - 1);

}

int main () {

   int num;

   cout << "Enter a number: ";

   cin >> num;

   cout << "Factorial of " << num << " is " << factorial(num) <<
endl;

   return 0;

}
```

Practice Question: Implement a recursive function for computing the Fibonacci series up to **n** terms.

1.4.4 Lab 4: Arrays and Strings

Objective: Learn array manipulations and basic string operations.

1. Write a program to find the largest element in an array.

Code Example:

```
#include <iostream>
using namespace std;
int main () {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int arr[n]; // Declare an array of size n
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    int max = arr[0];
    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    cout << "Largest element is " << max << endl;
    return 0;
}
```

Practice Question: Write a program to reverse a string.

1.4.5 Lab 5: Pointers and Dynamic Memory

Objective: Understand pointers, dynamic memory allocation, and pointer arithmetic.

1. Write a program to allocate memory dynamically for an array and find its sum.

Code Example:

```
#include <iostream>

using namespace std;

int main () {

    int n;

    cout << "Enter the number of elements: ";

    cin >> n;

    // Dynamically allocate memory for the array

    int* arr = new int[n];

    cout << "Enter " << n << " elements: ";

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

        cin >> arr[i];

    }

    int sum = 0;

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

        sum += arr[i];

    }
    cout << "Sum of the elements is " << sum << endl;

    // Deallocate the memory

    delete [] arr;

    return 0;

}
```

Practice Question: Implement pointer arithmetic to traverse an array.

1.4.6 Lab 6: Classes and Objects

Objective: Learn the concepts of object-oriented programming, including classes and objects.

1. Create a class **Rectangle** with length and breadth as data members, and methods to calculate area and perimeter.

Code Example:

```
#include <iostream>

using namespace std;

class Rectangle {

private:

    double length;

    double breadth;

public:

    // Constructor to initialize the rectangle dimensions

    Rectangle (double l, double b) {

        length = l;
        breadth = b;
    }
    // Method to calculate the area of the rectangle
    double area() {
        return length * breadth;
    }
    // Method to calculate the perimeter of the rectangle
    double perimeter() {
        return 2 * (length + breadth);
    }
    // Method to display the dimensions, area, and perimeter
    void display() {
        cout << "Length: " << length << endl;
        cout << "Breadth: " << breadth << endl;
        cout << "Area: " << area() << endl;
        cout << "Perimeter: " << perimeter() << endl;
    }
};
int main() {
    double length, breadth;
    cout << "Enter the length of the rectangle: ";
    cin >> length;
    cout << "Enter the breadth of the rectangle: ";
    cin >> breadth;
    // Create a Rectangle object
    Rectangle rect(length, breadth);
    // Display the dimensions, area, and perimeter
    rect.display();
    return 0;
}
```

Practice Question: Create a class Circle and find the area and the Perimeter of the Circle.

1.4.7 Lab 7: Operator Overloading

Objective: Understand operator overloading in C++. **Tasks**:

1. Overload the **+** operator to add two complex numbers using a class **Complex**.

Code Example:

```
#include <iostream>
using namespace std;
class Complex {
private:
    double real;
    double image;
public:
    // Constructor to initialize the complex number
    Complex(double r = 0.0, double i = 0.0): real(r), image(i) {}
    // Overload the + operator to add two complex numbers
    Complex operator + (const Complex& other) const {
        return Complex(real + other.real, image + other.image);
    }
    // Method to display the complex number
    void display() const {
        cout << real << " + " << image << "i" << endl;
    }
};
int main() {
    double real1, imag1, real2, imag2;
    cout << "Enter the real and imaginary parts of the first complex
number: ";
    cin >> real1 >> imag1;
    cout << "Enter the real and imaginary parts of the second
complex number: ";
    cin >> real2 >> imag2;
    // Create two Complex objects
    Complex c1(real1, imag1);
    Complex c2(real2, imag2);
    // Add the two complex numbers using the overloaded + operator

    Complex c3 = c1 + c2;

    // Display the result

    cout << "Sum of the two complex numbers: ";

    c3.display();

    return 0;

}
```

Practice Question: Overload the **<<** operator for outputting the complex number.

1.4.8 Lab 8: Inheritance and Polymorphism

Objective: Explore inheritance and polymorphism in C++. **Tasks**:

1. Implement a base class **Shape** and derive classes **Circle** and **Rectangle**.

Code Example:

```
#include <iostream>

#include <cmath> // For M_PI

using namespace std;

// Base class Shape

class Shape {

public:

    virtual double area() const = 0;    // Pure virtual function for area
    virtual double perimeter() const = 0; // Pure virtual function for
perimeter
    virtual ~Shape() {} // Virtual destructor
};
// Derived class Circle
class Circle: public Shape {
private:
    double radius;
public:
    Circle(double r): radius(r) {}
    double area() const override {
        return M_PI * radius * radius;
    }
    double perimeter() const override {
        return 2 * M_PI * radius;
    }
    void display() const {
        cout << "Circle: " << endl;
        cout << "Radius: " << radius << endl;
        cout << "Area: " << area() << endl;
        cout << "Perimeter: " << perimeter() << endl;
    }
```

```
};

// Derived class Rectangle

class Rectangle : public Shape {

private:

    double length;

    double breadth;

public:

    Rectangle(double l, double b) : length(l), breadth(b) {}

    double area() const override {

        return length * breadth;

    }

    double perimeter() const override {

        return 2 * (length + breadth);

    }

    void display() const {

        cout << "Rectangle: " << endl;

        cout << "Length: " << length << endl;

        cout << "Breadth: " << breadth << endl;

        cout << "Area: " << area() << endl;

        cout << "Perimeter: " << perimeter() << endl;

    }

};
```

```
int main() {
    double radius, length, breadth;
    // Input and create Circle object
    cout << "Enter the radius of the circle: ";
    cin >> radius;
    Circle circle(radius);
    // Input and create a Rectangle object
    cout << "Enter the length and breadth of the rectangle: ";
    cin >> length >> breadth;
    Rectangle rectangle(length, breadth);
    // Display details of Circle
    circle.display();
    // Display details of Rectangle
    rectangle.display();
    return 0;
}
```

Practice Question: Demonstrate polymorphism using virtual functions to calculate area.

1.4.9 Lab 9: Linked Lists

Objective: Implement and manipulate linked lists. **Tasks**:

1. Create a singly linked list and perform insertions and deletions.

Code Example:

```
#include <iostream>

using namespace std;

class Node {

public:

  int data;

  Node* next;

    Node(int data) {

    this->data = data;

    this->next = nullptr;

  }

};

class SinglyLinkedList {

private:

  Node* head;

  public:

  SinglyLinkedList() {

    head = nullptr;

  }

    // Function to insert a node at the beginning

  void insertAtBeginning(int data) {

    Node* newNode = new Node(data);
    newNode->next = head;

    head = newNode;

}

    // Function to insert a node at the end

void insertAtEnd(int data) {

    Node* newNode = new Node(data);

    if (head == nullptr) {

        head = newNode;

        return;

    }

    Node* temp = head;

    while (temp->next != nullptr) {

        temp = temp->next;

    }

    temp->next = newNode;

}

    // Function to delete a node by value

void deleteByValue(int data) {

    if (head == nullptr) {

        cout << "List is empty." << endl;

        return;

    }
```

```
        if (head->data == data) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr && temp->next->data != data) {
            temp = temp->next;
        }
        if (temp->next == nullptr) {
            cout << "Node with value " << data << " not found." << endl;
            return;
        }
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        delete nodeToDelete;
    }
        // Function to display the list
    void display() {
        if (head == nullptr) {
            cout << "List is empty." << endl;
            return;
        }
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "nullptr" << endl;
    }
        // Destructor to free the allocated memory
    ~SinglyLinkedList() {
        Node* temp = head;
        while (temp != nullptr) {
            Node* next = temp->next;
            delete temp;
            temp = next;
        }
    }
};
int main() {
    SinglyLinkedList list;
```

```
int choice, value;

while (true) {

    cout << "\nMenu:\n";

    cout << "1. Insert at Beginning\n";

    cout << "2. Insert at End\n";

    cout << "3. Delete by Value\n";

    cout << "4. Display List\n";

    cout << "5. Exit\n";

    cout << "Enter your choice: ";

    cin >> choice;

    switch (choice) {

        case 1:

            cout << "Enter value to insert at the beginning: ";

            cin >> value;

            list.insertAtBeginning(value);

            break;

        case 2:

            cout << "Enter value to insert at the end: ";

            cin >> value;

            list.insertAtEnd(value);

            break;

        case 3:
                cout << "Enter value to delete: ";
                cin >> value;
                list.deleteByValue(value);
                break;
            case 4:
                list.display();
                break;
            case 5:
                cout << "Exiting..." << endl;
                return 0;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }
    return 0;
}
```

Practice Question: Traverse the linked list and display its elements.

1.4.10 Lab 10: Stacks and Queues

Objective: Implement stack and queue data structures using arrays and linked lists.

1. Implement a stack with push and pop operations.

Code Example:

```
#include <iostream>

using namespace std;

class Node {

public:

    int data;

    Node* next;

        Node(int data) {

        this->data = data;

        this->next = nullptr;

    }

};

class Stack {

private:

    Node* top;

public:

    Stack() {

        top = nullptr;

    }

    // Function to push an element onto the stack

    void push(int data) {

        Node* newNode = new Node(data);
        newNode->next = top;

        top = newNode;

    }

    // Function to pop an element from the stack

    int pop() {

        if (top == nullptr) {

            cout << "Stack underflow. Cannot pop from an empty
stack." << endl;

            return -1;

        }

        Node* temp = top;

        top = top->next;

        int poppedData = temp->data;

        delete temp;

        return poppedData;

    }

    // Function to display the stack

    void display() {

        if (top == nullptr) {

            cout << "Stack is empty." << endl;

            return;

        }
        Node* temp = top;
        while (temp != nullptr) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "nullptr" << endl;
    }
    // Destructor to free the allocated memory
    ~Stack() {
        while (top != nullptr) {
            Node* temp = top;
            top = top->next;
            delete temp;
        }
    }
};
int main() {
    Stack stack;
    int choice, value;
    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Push\n";
cout << "2. Pop\n";

cout << "3. Display Stack\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

    case 1:

        cout << "Enter value to push: ";

        cin >> value;

        stack.push(value);

        break;

    case 2:

        value = stack.pop();

        if (value != -1) {

            cout << "Popped value: " << value << endl;

        }

        break;

    case 3:

        stack.display();

        break;

    case 4:

        cout << "Exiting..." << endl;
```

```
                return 0;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    }
    return 0;
}
```

Practice Question: Implement a queue with enqueue and dequeue operations.

1.4.11 Lab 11: Sorting and Searching Algorithms

Objective: Implement common sorting and searching algorithms.

1. Implement Bubble Sort.

Code Example:

```
#include <iostream>
using namespace std;
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // Last i elements are already in place
        for (int j = 0; j < n - i - 1; j++) {
            // Swap if the element found is greater than the next element
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}
int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
        cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
        bubbleSort(arr, n);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Practice Question:

1. Implement Binary Search.
2. Implement Insertion sort.
3. Implement Merge Sort.

1.5 SUMMARY

The C++ Programming Lab Manual is designed to provide students with practical experience in programming concepts using the C++ language. It covers a wide range of topics, including basic syntax, control flow, data structures, and algorithms. Through hands-on exercises, students will gain proficiency in C++ programming, develop problem-solving skills, and acquire a solid understanding of fundamental data structures and algorithms. The lab manual aims to prepare students for real-world application development by providing practical experience in implementing C++ programs for various scenarios and applications.

1.6 QUESTIONS

1. Write a program to find the sum of two numbers.
2. Implement a program to check whether a given number is even or odd.
3. Write a program to find the factorial of a given number.
4. Implement a program to swap two numbers without using a temporary variable.

5. Write a program to check if a given year is a leap year or not.
6. Implement a stack using an array.
7. Write a program to reverse a linked list.
8. Implement a recursive function to find the nth Fibonacci number.
9. Write a program to implement binary search in a sorted array.
10. Implement the Depth First Search (DFS) algorithm for a graph.
11. Write a program to find the largest element in an array.
12. Implement a function to check if a given string is a palindrome.

References

- Bjarne Stroustrup, "The C++ Programming Language."
- E. Balagurusamy, "Object Oriented Programming with C++."
- Robert Lafore, "Data Structures and Algorithms in C++."
- Mark Allen Weiss, "Data Structures and Algorithm Analysis in C++."