Annexure-I

"Browser History Management System using C++"

Cse Pathshala

A training report

Submitted in partial fulfillment of the requirements for the award of degree of
Bachelor of Computer Science and Engineering with Ai-ML

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

SUBMITTED BY

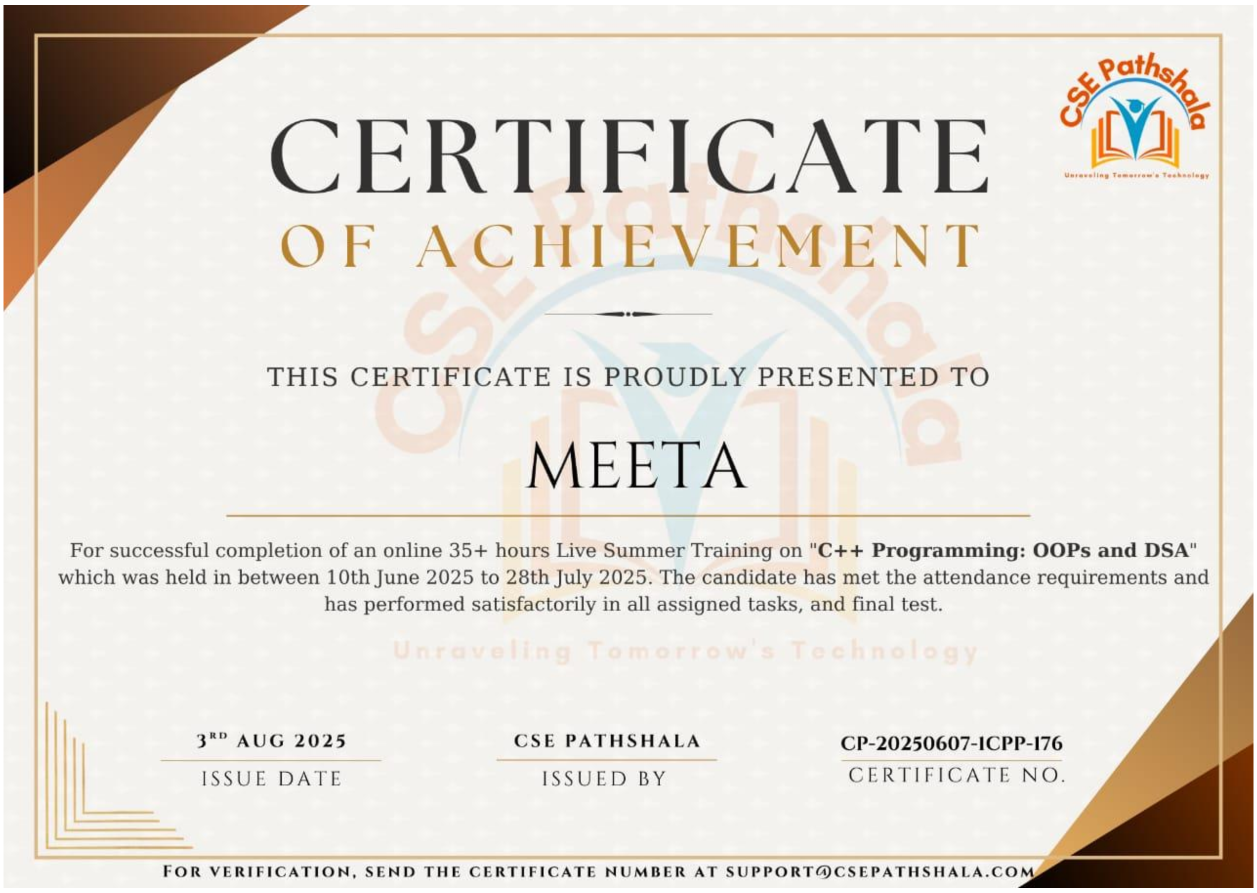
Name of student: Meeta

Registration Number:12311739

Annexure-II: Student Declaration

To whom so ever it may concern

I, Meeta, 12311739, hereby declare that the work done by me on "Browser History Management System using C++"" for period June 2025 to July 2025, is a record of original work for the partial fulfillment of the requirements for the award of the degree Bachelor of Computer Science and Engineering with Ai-ML.



CERTIFICATE

OF ACHIEVEMENT

THIS CERTIFICATE IS PROUDLY PRESENTED TO

MEETA

For successful completion of an online 35+ hours Live Summer Training on "**C++ Programming: OOPs and DSA**" which was held in between 10th June 2025 to 28th July 2025. The candidate has met the attendance requirements and has performed satisfactorily in all assigned tasks, and final test.

3RD AUG 2025	CSE PATHSHALA	CP-20250607-ICPP-176
ISSUE DATE	ISSUED BY	CERTIFICATE NO.

FOR VERIFICATION, SEND THE CERTIFICATE NUMBER AT SUPPORT@CSEPATHSHALA.COM

Name of the Student Meeta
12311739

ACKNOWLEDGEMENT

am pleased to take this opportunity to express my heartfelt gratitude to everyone who supported and guided me throughout the course of my summer internship project titled: "Browser History Management System using C++"
First and foremost, I extend my sincere thanks to the faculty at CSE Pathshala for providing me with this opportunity and for their expert guidance throughout the internship. Their insightful feedback, consistent encouragement, and technical mentorship were instrumental in shaping the progress and successful completion of this project.
I would also like to express my gratitude to the faculty and staff of Lovely Professional University for equipping me with the academic foundation and resources that made this project possible. Their constant support and motivation have been invaluable.
Special thanks to my peers and classmates, whose collaboration and constructive suggestions helped me refine the system and improve its effectiveness.
Lastly, I am deeply thankful to my family and friends for their unwavering support, patience, and motivation during this internship. Their belief in me inspired confidence and focus throughout the project.
This internship, conducted under CSE Pathshala, not only strengthened my technical proficiency in C++ programming, data structures, and file handling but also provided meaningful exposure to building real-world systems from scratch. I am truly grateful to everyone who contributed to the success of this learning journey.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

As part of the Summer Training Program (June–July 2025) organized by CSE Pathshala, I had the opportunity to participate in an intensive online course titled "C++ Programming: OOPs and DSA." The training consisted of over 35+ hours of live interactive sessions designed to bridge the gap between theoretical learning and practical application.
Known for its commitment to quality technical education, CSE Pathshala provided a structured and hands-on learning experience. The program emphasized core programming concepts, object-oriented programming (OOPs), and foundational as well as advanced topics in Data Structures and Algorithms (DSA).
The course covered a range of topics including:

- **Object-Oriented Programming Principles – encapsulation, inheritance, polymorphism, and abstraction.**
- **Essential Data Structures – arrays, linked lists, stacks, queues, trees, graphs.**
- **Algorithms – searching, sorting, recursion, dynamic programming, and greedy algorithms.**
- **Problem Solving Techniques – through live coding, assignments, and tests.**

Throughout the training, I actively engaged in practical exercises and coding challenges using C++, improving my logic-building skills, algorithmic thinking, and coding efficiency.
CSE Pathshala's learning environment, mentorship, and well-structured curriculum enabled me to develop a deeper understanding of computer science fundamentals. The program not only strengthened my technical foundation in C++ and DSA but also instilled valuable soft skills such as problem-solving, time management, and independent learning.
Successfully completing the final evaluation and fulfilling attendance criteria, I received a Certificate of Achievement acknowledging my performance and dedication throughout the program.
This training experience has significantly enriched my academic profile and prepared me for future challenges in software development and technical interviews.
Top of Form

1.1 OVERVIEW OF TRAINING DOMAIN

In the evolving landscape of software development, mastering programming fundamentals and algorithmic thinking is essential for building efficient, scalable, and maintainable solutions. During my summer training (June–July 2025) under the program titled "C++ Programming: OOPs and DSA" offered by CSE Pathshala, I deepened my understanding of these core technical domains through both theoretical learning and practical implementation.

This training emphasized two foundational pillars of computer science:

- **Object-Oriented Programming (OOPs) – a paradigm focused on structuring code using objects and classes to promote modularity, reusability, and clarity.**
- **Data Structures and Algorithms (DSA) – the backbone of efficient programming,**

crucial for solving complex problems and optimizing performance.

To solidify my understanding, I worked on practical coding problems and developed console-based applications in C++, including a mini-project like a Browser History Manager that simulated stack-based navigation of visited web pages. This involved implementing:

- **Stack data structure for managing browser history.**
- **Concepts like encapsulation, class abstraction, and method design.**
- **Menu-driven interfaces using loops and conditional statements.**

The training covered critical areas such as:

- **Class and Object Design – applying OOP principles like inheritance, encapsulation, and polymorphism.**
- **Core Data Structures – including arrays, stacks, queues, linked lists, trees, and graphs.**
- **Algorithmic Techniques – recursion, sorting, searching, and greedy approaches.**
- **Time and Space Complexity – understanding the trade-offs of different algorithms.**

This domain also required strong logical reasoning and problem-solving capabilities, essential for:

- **Optimizing code for performance.**
- **Debugging and maintaining clean code structures.**
- **Preparing for competitive programming and technical interviews.**

The training enhanced my skills not only in C++ syntax but also in developing efficient and modular programs. It bridged the gap between academic concepts and practical software development, equipping me with the confidence and capabilities to tackle real-world coding challenges.

By the end of the program, I was able to demonstrate my grasp of C++ and DSA through regular assignments, hands-on projects, and a final evaluation—earning a Certificate of Achievement from CSE Pathshala. This training experience laid a strong foundation for advanced programming and future software engineering role.

1.2 OBJECTIVE OF THE PROJECT

The main objective of my summer training project, conducted as part of the "C++ Programming: OOPs and DSA" course by CSE Pathshala, was to design and implement a console-based application using C++ that demonstrates the application of object-oriented programming principles and core data structures.
The specific focus of my project was to build a Browser History Manager, simulating real-world browser navigation using a stack to manage visited pages. The primary aim was to apply theoretical concepts of OOP and DSA in a practical and intuitive way to develop a system with the following capabilities:

Project Objectives:
1. Functional Objectives:
- **To simulate visiting web pages and storing browsing history using a stack-based approach.**
- **To implement "Go Back" functionality, similar to real-world browsers, using stack pop operations.**
- **To display the current active page using peek-like operations on the stack.**
- **To provide a menu-driven interface allowing users to interact with the application in a loop-based format.**

2. Technical Objectives:
- **To apply key Object-Oriented Programming (OOP) concepts such as:**
  - **Encapsulation – by defining a BrowserHistory class with private data members and public methods.**
  - **Abstraction – by hiding internal stack logic and exposing only relevant user operations.**
- **To utilize the Stack data structure to manage browsing history efficiently.**
- **To implement array-based stack operations, including:**
  - **push() (visiting a new page),**
  - **pop() (going back to the previous page),**
  - **peek() (viewing the current page).**
- **To handle edge cases such as stack overflow (history full) and underflow (no previous pages).**

3. Broader Learning Objectives:
- **To gain hands-on experience in writing modular and efficient C++ code.**
- **To strengthen algorithmic thinking through stack manipulation and control flow logic.**
- **To understand the relationship between data structures and real-world applications (e.g., how a browser handles navigation history).**
- **To reinforce theoretical knowledge gained in the training through the design, implementation, and testing of a complete C++ project.**

By the end of the project, I successfully demonstrated my understanding of OOP and DSA by implementing a fully functional and user-interactive console application. This experience not only enhanced my coding skills but also prepared me for more complex

programming challenges and real-world software development.

1.3 RELEVANCE TO REAL-WORLD SCENARIOS

The practical relevance of this project lies in its simulation of a real-world browser navigation system, showcasing how data structures and object-oriented programming (OOP) can be effectively applied to solve user-centric problems in software development.

In today's digital environment, users interact with web browsers constantly, and behind every browser lies a mechanism to manage navigation history efficiently. The Browser History Manager project replicates this fundamental component by applying stack-based logic to mimic the "visit", "go back", and "current page" functionalities of modern browsers.

---

1. Real-World Relevance in Software Applications

a. Web Browsers:

Modern browsers like Chrome, Firefox, and Edge use stacks to track pages visited by the user.

When a user clicks "Back," the browser simply pops the last visited page off the stack and displays the previous one — precisely what this project simulates.

This mirrors real-time functionality, demonstrating how abstract data structures like stacks can power intuitive and essential user experiences.

b. System Navigation & Undo/Redo Features:

- **Beyond browsers, stack-based logic is central to undo-redo operations in text editors, IDEs, and graphic design tools.**
- **Operating systems and mobile apps use similar principles for activity stack management, allowing seamless transitions between screens or states.**

---

2. Technical Learning with Practical Impact

By implementing this system using C++, the project:

- **Connects theoretical data structures to practical software components.**
- **Strengthens understanding of memory-efficient history management.**
- **Demonstrates how OOP (via class design and encapsulation) can create modular, maintainable code.**
- **Prepares learners for more advanced system design by reinforcing logic building, abstraction, and control flow.**

---

3. Broader Implications

While the application developed is small-scale, the underlying logic is scalable and adaptable for:

Real-time system development (e.g., browser engines, session management).

Interview preparation through hands-on problem-solving using stacks and OOP.

Future extensions such as forward navigation, bookmarking, tab management, or graphical UI integration.

Through this project, I gained insight into how fundamental programming concepts are

directly relevant to real-world software systems, making it not only a valuable academic exercise but also a practical stepping stone toward becoming an industry-ready developer.

CHAPTER:2 TRAINING OVERVIEW

2.1 TOOLS AND TECHNOLOGIES USED

To design, implement, and test the Browser History Manager project, I utilized a set of tools and technologies commonly used in system-level programming and algorithm development. These tools were instrumental in applying object-oriented programming (OOP) concepts and data structures like stacks in a real-world context.

---

1. C++ Programming Language

- **C++ was the core language used for the project.**
- **Its support for OOP, manual memory management, and performance optimization made it ideal for implementing low-level logic and building efficient system simulations.**
- **Features like classes, arrays, and control structures enabled the development of a structured, reusable, and interactive application.**

---

2. Object-Oriented Programming (OOP)

- **Core OOP concepts such as encapsulation, abstraction, and method definition were applied through a Browser History class.**
- **The class encapsulated a stack-based history mechanism, exposing clean interfaces for operations like visiting a new page, going back, and checking the current page.**

---

3. Stack Data Structure (Array-Based)

- **A fixed-size array was used to implement the stack that stored browsing history.**
- **Stack operations like push, pop, and peek were manually coded, reinforcing my understanding of how data structures function internally.**
- **Stack overflow and underflow conditions were handled gracefully using conditional logic.**

---

4. Code::Blocks / Visual Studio Code (IDE)

- **Code was developed and tested using modern IDEs such as Code::Blocks or Visual Studio Code, which provided:**
  - **Syntax highlighting**

- Debugging support
- Real-time error checking
- Streamlined compilation and execution of C++ code

---

5. GCC/G++ Compiler

- **The project was compiled using the GNU Compiler Collection (GCC/G++), ensuring standard-compliant C++ code execution.**
- **The compiler facilitated efficient error detection and optimized binary generation for the application.**

---

6. Command-Line Interface (CLI)

- **A menu-driven CLI was designed using standard I/O streams (cin, cout) to interact with the user.**
- **This offered an intuitive interface to simulate browser navigation, test stack behavior, and verify application logic without a GUI.**

This stack of tools and technologies allowed me to focus on building a robust backend logic, improve my C++ coding proficiency, and strengthen my understanding of how data structures drive real-world applications like browser navigation systems. The experience also laid the foundation for more advanced system development in the future.

2.2 AREAS COVERED DURING TRAINING

During the summer training organized by CSE Pathshala on "C++ Programming: OOPs and DSA", we explored a wide range of topics designed to strengthen our foundation in structured and object-oriented programming, as well as algorithmic problem-solving. The training combined theoretical learning with practical implementation, culminating in the development of projects like a Browser History Manager.

---

1. Fundamentals of C++ Programming
We began by revisiting and strengthening core C++ programming concepts such as:

- **Variables, data types, operators, and expressions**
- **Conditional statements and loops (e.g., if, switch, for, while)**
- **Functions, parameters, and return values**
- **Input/output operations using cin and cout**

---

2. Object-Oriented Programming (OOP) in C++
We explored the principles and real-world applications of OOP, including:

- **Classes and Objects**
- **Encapsulation – bundling data with methods**
- **Abstraction – exposing only relevant details to users**

- **Inheritance and Polymorphism – for code reusability and flexibility (covered in broader scope)**
  These concepts were applied directly in our project to structure clean and modular code.

---

3. Data Structures
A major portion of the training was devoted to the study and implementation of fundamental data structures using C++. Key topics included:

- **Arrays and Strings**
- **Stacks and Queues**
- **Linked Lists (Singly and Doubly)**
- **Trees and Graphs (introduction)**
- **Recursion and its application in DSA**

In the Browser History Manager project, we implemented a stack using arrays to simulate back navigation in browsers — demonstrating the real-world use of this data structure.

---

4. Algorithmic Thinking
We covered various algorithmic techniques to solve problems efficiently:

- **Searching algorithms (Linear Search, Binary Search)**
- **Sorting algorithms (Bubble Sort, Selection Sort, Insertion Sort, etc.)**
- **Time and Space Complexity analysis to evaluate the efficiency of different algorithms**

---

5. Problem-Solving and Code Implementation
Throughout the training, we:

- **Practiced solving structured and unstructured problems using C++**
- **Developed logical flow for real-world applications**
- **Understood best practices for writing efficient and readable code**

---

6. Hands-on Project Development
To reinforce theoretical knowledge, we implemented real-time, console-based applications like:

- **A Browser History Manager using stack operations in C++**
- **Other small-scale programs applying DSA and OOP principles**

These projects helped bridge the gap between academic understanding and practical programming skills.

This training equipped me with a solid foundation in both programming methodology and data structures, preparing me for further challenges in software development, coding interviews, and advanced computing topics. The hands-on exposure, particularly through project work, played a vital role in transforming abstract concepts into tangible coding solutions.

2.3 DAILY/WEEKLY WORK SUMMARY

he summer training program on “C++ Programming: OOPs and DSA”, conducted by CSE Pathshala, was organized in a well-structured, module-based format spread across ten weeks. The training combined theory sessions, coding practice, and a capstone project

to reinforce real-world programming concepts.

Week 1: Introduction to C++ and Programming Fundamentals

- **Overview of C++ syntax, structure, and execution flow.**
- **Practice on variables, data types, operators, and basic I/O.**
- **Introduction to development tools (e.g., Code::Blocks, VS Code, GCC compiler).**

Week 2: Control Structures and Functions

- **Implementation of control flow: if, else, switch, while, for, and do-while.**
- **Introduction to modular programming with functions.**
- **Scope of variables, function overloading, and return types.**

Week 3: Object-Oriented Programming Concepts

- **Understanding core OOP principles: classes, objects, encapsulation.**
- **Writing and testing class definitions with access specifiers (public, private).**
- **Constructor and destructor implementation.**

Week 4: Arrays and String Handling

- **One-dimensional and two-dimensional arrays in C++.**
- **Basic string manipulation using char arrays and string class.**
- **Implementation of simple array-based operations and programs.**

Week 5: Stack and Queue Data Structures

- **Introduction to linear data structures: stack and queue.**
- **Manual implementation using arrays.**
- **Stack operations: push(), pop(), peek(), and overflow/underflow conditions.**
- **Start of project planning (Browser History Manager).**

Week 6: Linked Lists and Dynamic Memory Allocation

- **Implementation of singly and doubly linked lists.**
- **Use of pointers and new/delete for dynamic memory.**
- **Comparison of static vs dynamic structures.**

Week 7: Trees and Recursive Algorithms

- **Introduction to binary trees and traversal methods (inorder, preorder, postorder).**
- **Recursion and its use in tree operations and sorting algorithms.**
- **Practice on recursive vs iterative logic.**

Week 8: Sorting, Searching, and Algorithm Analysis

- **Implementation of sorting algorithms: Bubble Sort, Insertion Sort, Selection Sort.**
- **Searching techniques: Linear and Binary Search.**
- **Introduction to time and space complexity (Big-O notation).**

Week 9: Project Development – Browser History Manager

- **Design and coding of the BrowserHistory class using a stack.**
- **Implemented features:**
  - **Visit new page (push)**
  - **Go back (pop)**
  - **Show current page (peek)**
- **Developed a menu-driven interface to simulate browser navigation.**

Week 10: Project Finalization and Evaluation

- **Debugging, edge-case handling, and user experience improvements.**
- **Documentation of the code and writing output test cases.**
- **Final evaluation based on project completion, coding style, and functionality.**
- **Issuance of Certificate of Achievement upon successful completion.**

This structured training plan helped build a strong foundation in C++ programming, OOP, and data structures, culminating in a practical application that mirrors a real-world use case. Each week progressively deepened my understanding, leading to a project that demonstrated my skills in problem-solving, code structuring, and logic implementation.

CHAPTER 3: PROJECT DETAILS

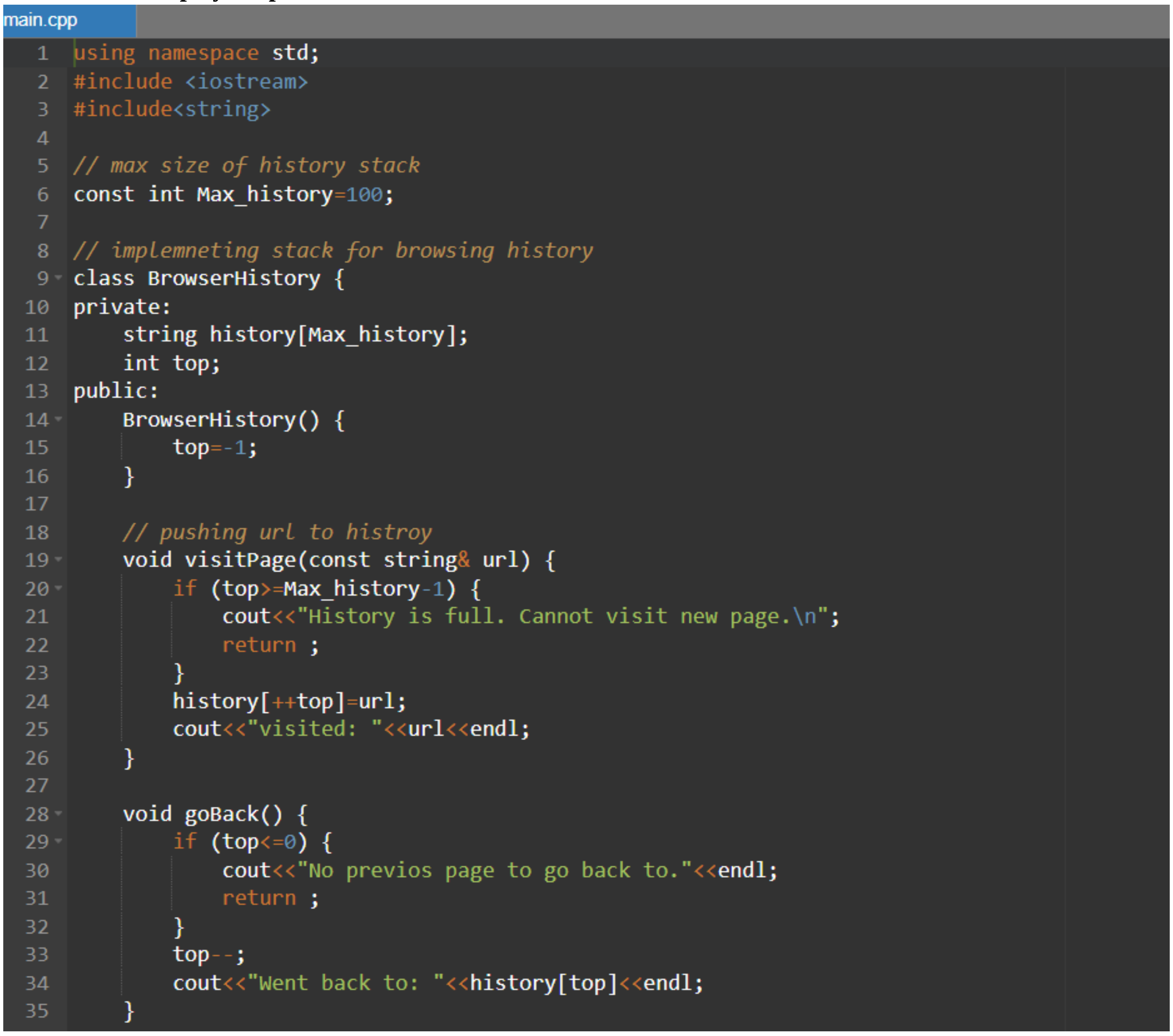
3.1 TITLE OF THE PROJECT

"Efficient C++ Applications using OOP and Data Structures"

Project Overview

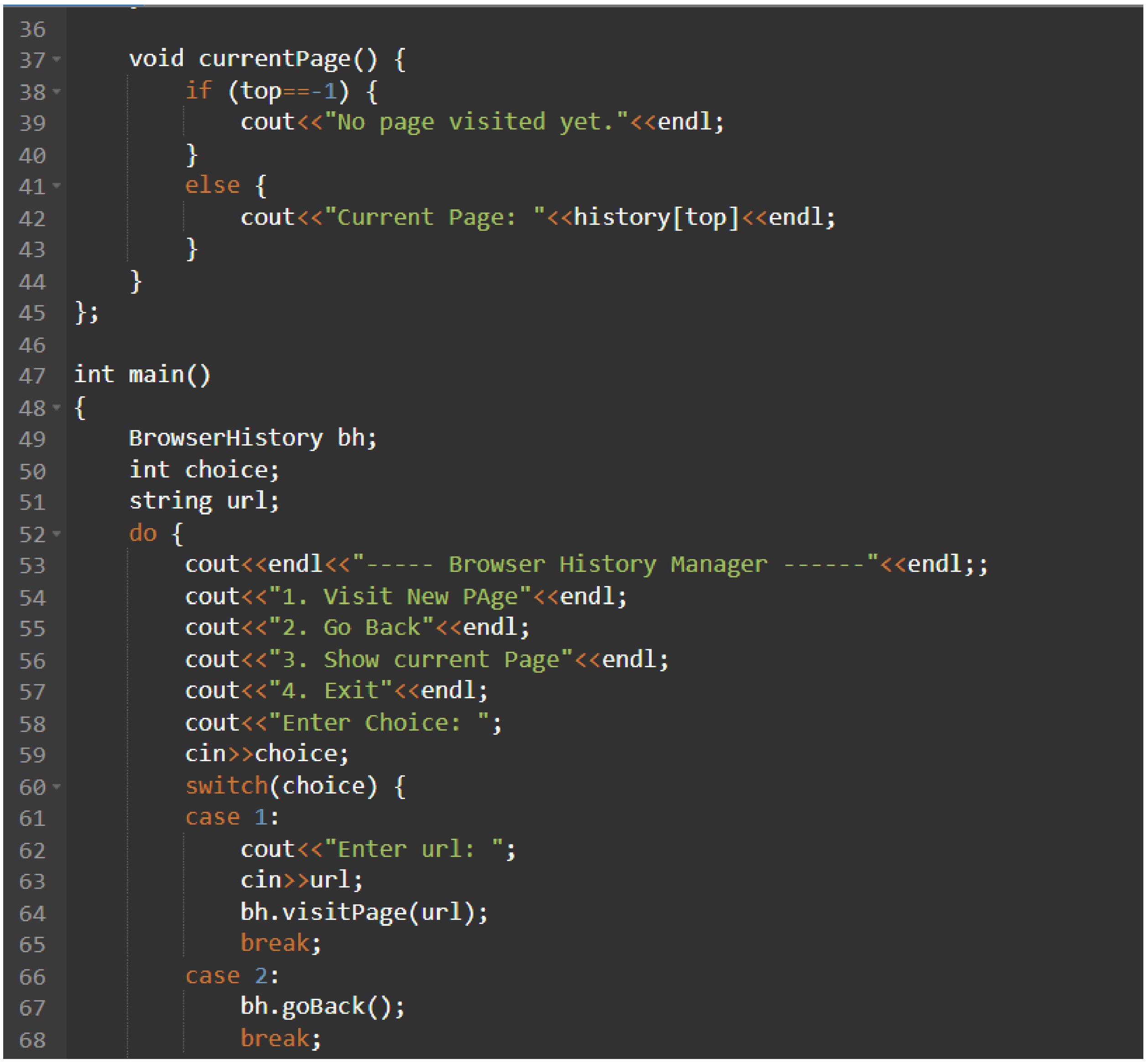
The aim of this project was to build a set of efficient and modular C++ applications using core Object-Oriented Programming (OOP) principles and fundamental data structures. These programs demonstrated real-world problem-solving capabilities using C++ with a focus on:

- **Class and Object design**
- **File handling operations**
- **Use of data structures like stacks, queues, and linked lists**
- **Inheritance and polymorphism for reusable and scalable code**

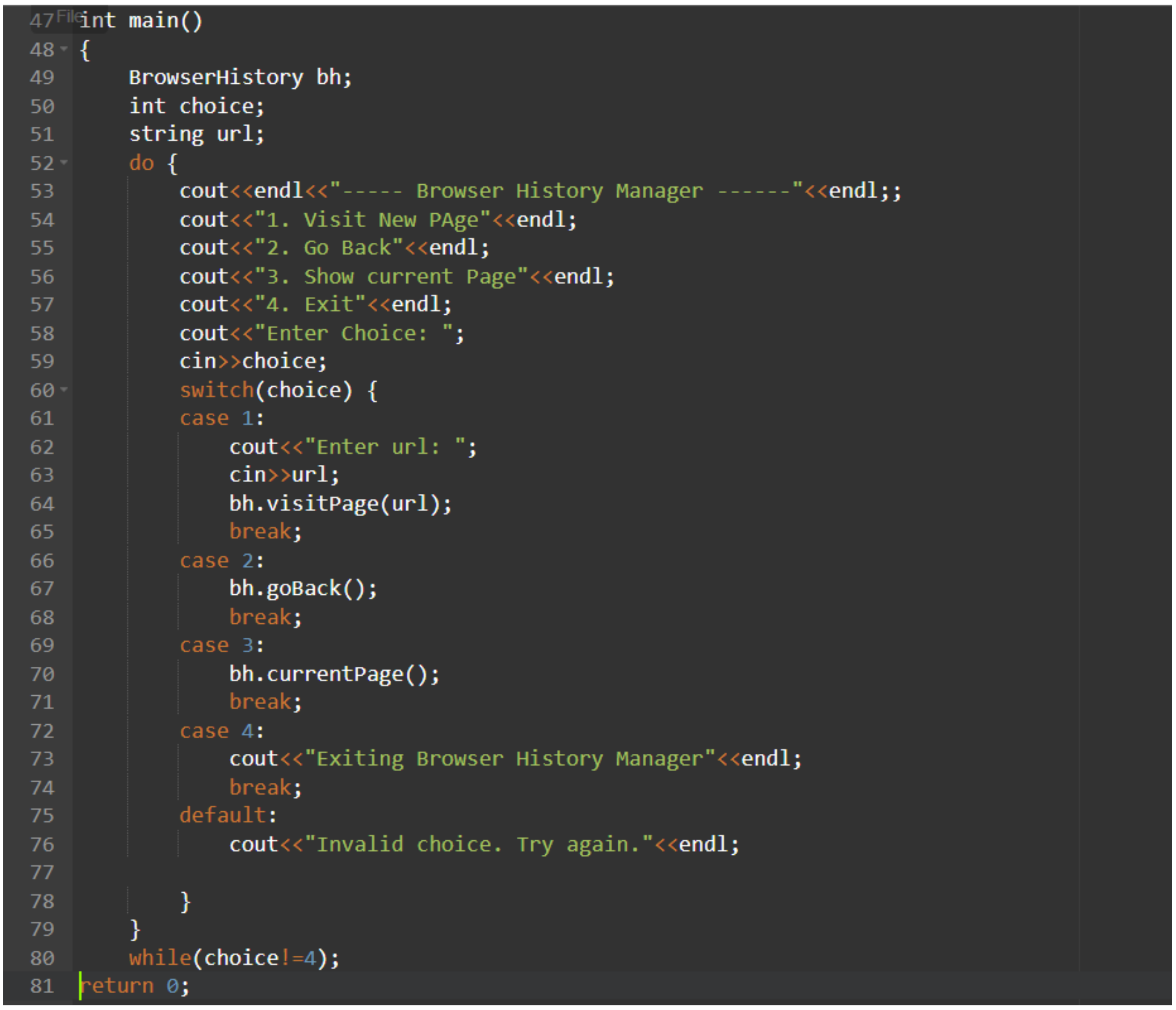


main.cpp

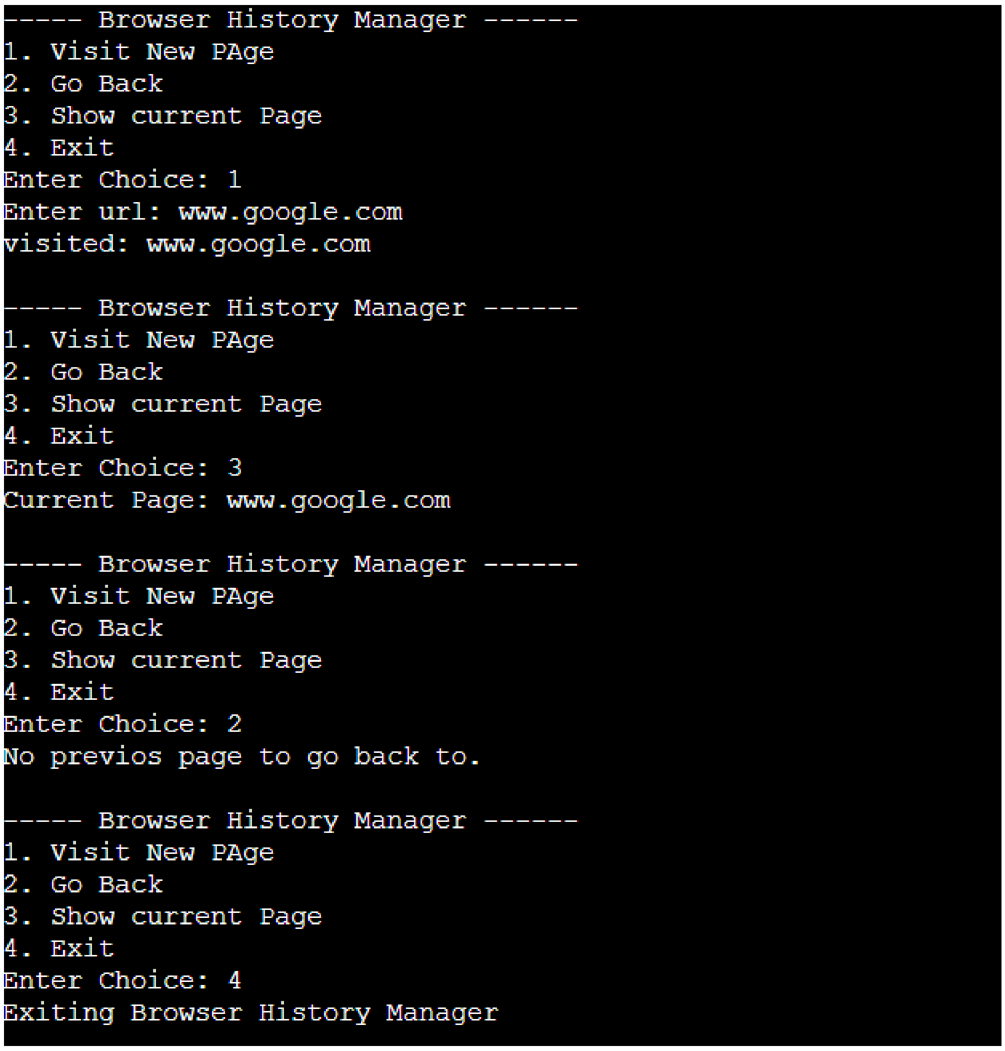
```cpp
using namespace std;
#include <iostream>
#include<string>

// max size of history stack
const int Max_history=100;

// implemneting stack for browsing history
class BrowserHistory {
private:
    string history[Max_history];
    int top;
public:
    BrowserHistory() {
        top=-1;
    }

    // pushing url to histroy
    void visitPage(const string& url) {
        if (top>=Max_history-1) {
            cout<<"History is full. Cannot visit new page.\n";
            return ;
        }
        history[++top]=url;
        cout<<"visited: "<<url<<endl;
    }

    void goBack() {
        if (top<=0) {
            cout<<"No previos page to go back to."<<endl;
            return ;
        }
        top--;
        cout<<"Went back to: "<<history[top]<<endl;
    }
```



```cpp

    void currentPage() {
        if (top==-1) {
            cout<<"No page visited yet."<<endl;
        }
        else {
            cout<<"Current Page: "<<history[top]<<endl;
        }
    }
};

int main()
{
    BrowserHistory bh;
    int choice;
    string url;
    do {
        cout<<endl<<"----- Browser History Manager ------"<<endl;;
        cout<<"1. Visit New PAge"<<endl;
        cout<<"2. Go Back"<<endl;
        cout<<"3. Show current Page"<<endl;
        cout<<"4. Exit"<<endl;
        cout<<"Enter Choice: ";
        cin>>choice;
        switch(choice) {
        case 1:
            cout<<"Enter url: ";
            cin>>url;
            bh.visitPage(url);
            break;
        case 2:
            bh.goBack();
            break;
```



```cpp
int main()
{
    BrowserHistory bh;
    int choice;
    string url;
    do {
        cout<<endl<<"----- Browser History Manager ------"<<endl;;
        cout<<"1. Visit New PAge"<<endl;
        cout<<"2. Go Back"<<endl;
        cout<<"3. Show current Page"<<endl;
        cout<<"4. Exit"<<endl;
        cout<<"Enter Choice: ";
        cin>>choice;
        switch(choice) {
        case 1:
            cout<<"Enter url: ";
            cin>>url;
            bh.visitPage(url);
            break;
        case 2:
            bh.goBack();
            break;
        case 3:
            bh.currentPage();
            break;
        case 4:
            cout<<"Exiting Browser History Manager"<<endl;
            break;
        default:
            cout<<"Invalid choice. Try again."<<endl;

        }
    }
    while(choice!=4);
return 0;
```


```
----- Browser History Manager ------
1. Visit New PAge
2. Go Back
3. Show current Page
4. Exit
Enter Choice: 1
Enter url: www.google.com
visited: www.google.com

----- Browser History Manager ------
1. Visit New PAge
2. Go Back
3. Show current Page
4. Exit
Enter Choice: 3
Current Page: www.google.com

----- Browser History Manager ------
1. Visit New PAge
2. Go Back
3. Show current Page
4. Exit
Enter Choice: 2
No previos page to go back to.

----- Browser History Manager ------
1. Visit New PAge
2. Go Back
3. Show current Page
4. Exit
Enter Choice: 4
Exiting Browser History Manager
```

Key Features of the Project

- **OOP Concepts: Implemented classes for various components such as Vehicle, Room, Clock, etc., using constructors, destructors, and method overloading.**
- **File Handling: Demonstrated how to read/write/append to files using streams (fstream).**
- **Data Structures: Developed programs to implement and manipulate stacks, queues, and binary search trees.**
- **Real-life Applications: Created small utility programs such as:**
    - **A quiz game using virtual functions and polymorphism**
    - **A text editor simulation using dynamic memory allocation**
    - **A sales tracker for stock and revenue management**

Technologies Used

- **Programming Language: C++**
- **Compiler/IDE: Code::Blocks / Dev-C++ / g++ (Linux)**
- **Concepts Applied: OOP, STL, File I/O, Dynamic Memory, Class Design**

3.1 PROBLEM DEFINITION

C++ remains a foundational language for building high-performance systems and applications. Despite the rise of newer languages, industries continue to rely on C++ for its speed, efficiency, and low-level control, especially in areas such as system software, game development, embedded systems, and real-time applications.

However, many aspiring developers face challenges in effectively utilizing C++ features such as object-oriented programming (OOP), memory management, and file handling for real-world problem-solving. This internship aimed to bridge that gap through a series of structured, hands-on projects that mimic real-life scenarios.
The problems we addressed include:

- **Managing browser history using a stack-based structure**
- **Simulating a sales system that tracks product revenue and stock depletion**
- **Dynamically creating and manipulating strings using memory allocation**
- **Performing file operations such as reading, appending, and displaying contents**
- **Implementing quiz and score systems using virtual functions and polymorphism**
- **Building small utilities to demonstrate constructors, destructors, copy constructors, etc.**

These projects provided exposure to:

- **Data structures like stacks, queues, and vectors**
- **OOP principles such as encapsulation, inheritance, and polymorphism**
- **File I/O for persistent storage**
- **Memory management using pointers and dynamic allocation**

Through these mini-projects and use-cases, we developed robust coding practices and a deeper understanding of how C++ is used to develop modular, maintainable, and efficient applications.

3.2 SCOPE AND OBJECTIVES

3.31 SCOPE OF THE PROJECT

The scope of this project spans the design, development, and implementation of multiple C++ programs that simulate real-world applications using core concepts of object-oriented programming (OOP), file handling, dynamic memory allocation, and data structures.
The internship focused on building modular and efficient C++ programs that address common software development challenges through practical use-cases. The main components of the project include:

1. **System Utilities and Simulators**
   - **Programs to simulate browser history using stacks**
   - **File-based applications that append and read data**
   - **Simulated quiz systems using virtual functions and inheritance**
2. **Object-Oriented Programming**
   - **Implementation of classes with constructors, destructors, and copy constructors**
   - **Demonstrations of inheritance, polymorphism, encapsulation, and abstraction**
   - **Real-world use-cases like Room area calculator, Clock formatter, and Vehicle details manager**
3. **Memory Management**
   - **Use of dynamic memory allocation (new, delete)**
   - **String manipulation using pointers**
   - **Programs that handle dynamic input up to 1000 characters**

4. **File Handling**
    - **Reading from and writing to text files**
    - **Appending data without overwriting**
    - **Displaying contents of files for verification**

These programs are designed to enhance understanding of C++ by focusing on both conceptual clarity and practical implementation. The scope also includes best coding practices, code reusability, and modularity — crucial for building scalable and maintainable systems in the software industry.
This project lays a solid foundation for transitioning into more advanced system-level programming, competitive coding, or backend development roles that leverage the power and efficiency of C++.

3.32 OBJECTIVES OF THE PROJECT

The project aimed to strengthen practical and theoretical knowledge of C++ programming by developing a series of modular applications based on real-world scenarios. The key objectives were:

- **TO UNDERSTAND AND APPLY CORE C++ CONCEPTS, including variables, data types, control structures, functions, and arrays to develop logical solutions for everyday problems.**
- **TO IMPLEMENT OBJECT-ORIENTED PROGRAMMING (OOP) principles such as encapsulation, inheritance, polymorphism, and abstraction using C++ classes and objects.**
- **TO PRACTICE MEMORY MANAGEMENT, including the use of dynamic memory allocation (`new` and `delete` operators) and pointer-based operations for efficient resource usage.**
- **TO PERFORM FILE HANDLING OPERATIONS, such as reading from, writing to, and appending text files using file streams to simulate real-world data storage and retrieval tasks.**
- **TO DEVELOP USER-INTERACTIVE PROGRAMS using modular code structures for better usability and maintainability.**
- **TO BUILD MINI PROJECTS such as:**
    - **Vehicle registration and display system**
    - **Room area comparison using copy constructors**
    - **Digital clock formatter with zero-padding**
    - **Stack-based search for key-divisible elements**
    - **File-based quiz scoring system using inheritance**
- **TO ENHANCE DEBUGGING, CODE OPTIMIZATION, AND PROBLEM-SOLVING SKILLS through hands-on programming challenges and structured assignments.**
- **.**

3.3 SYSTEM REQUIREMENTS

To design, implement, and test C++ programs efficiently during the internship, the following hardware and software specifications were required.

Hardware Requirements

- **Processor: Intel Core i3 or higher (Core i5/i7 recommended) for fast compilation and smooth execution of C++ programs.**
- **Memory (RAM): Minimum 4 GB RAM (8 GB preferred) to support IDEs and multiple console applications running simultaneously.**
- **Storage: At least 1–2 GB of free disk space for source files, executables, and backups.**
- **Display: Standard monitor resolution (1366×768 or higher) for a better coding and debugging experience.**

Software Requirements

- **Programming Language: C++17 (ISO/IEC standard)**
- **Compiler:**
  - **GCC (MinGW) for Windows or g++ for Linux**
  - **MSVC (Microsoft C++ compiler) on Visual Studio (optional)**
- **Operating System: Windows 10 (used in development), compatible with Linux distributions like Ubuntu for cross-platform execution.**
- **Development Environments:**
  - **Code::Blocks, Dev C++, or Turbo C++ for quick compilation and debugging**
  - **Visual Studio Code with C++ extensions for modern, modular code development**
- **Version Control (optional): Git for source code management (if working in a collaborative environment)**

Additional Libraries/Tools (if used):

- **Standard Template Library (STL): For using built-in data**

structures like vectors, stacks, queues, and maps.
- **Graph plotting tools (optional): For output visualization, where required.**

Project Codebase Includes:
- **Programs involving:**
  - **Class-based design using constructors and inheritance**
  - **File I/O handling for reading/writing structured data**
  - **Data structure applications like stacks and queues**
  - **Dynamic memory and pointer usage**

These requirements ensured a robust environment for developing, testing, and debugging C++ applications efficiently.

4.1 TOOLS USED

To design and implement programs during the internship, a robust set of tools and environments were used to streamline C++ development. These tools were selected for their support of modern C++ features, ease of debugging, and ability to handle object-oriented and data structure-based programs.

---

1. C++ Programming Language
C++ was the primary programming language used in the internship. It enabled the development of efficient, modular, and object-oriented solutions for various tasks including file handling, class design, dynamic memory management, and data structure implementations like stacks, queues, and linked lists.

---

2. Code::Blocks IDE
Code::Blocks served as the integrated development environment for writing, compiling, and debugging C++ code. It offers a lightweight interface with support for:
- **GCC or MinGW compilers**
- **Breakpoints and step-by-step debugging**
- **Syntax highlighting and project-based organization**

---

3. Turbo C++ / Dev C++ (Optional)
These lightweight IDEs were optionally used for quickly testing smaller C++ programs. Their fast compilation and simple UI made them suitable for console-based application development.

---

4. Visual Studio Code
VS Code was also used with extensions such as:
- **C++ IntelliSense for code suggestions and error detection**
- **Code Runner for running C++ snippets within the editor**
- **Git Integration for version control (optional)**

---

5. GCC (GNU Compiler Collection)
The GCC compiler, especially through the MinGW setup on Windows, was used to compile

standard-compliant C++17/14 code. It provided support for modern features like STL (Standard Template Library), lambda expressions, and auto keyword.

---

6. Standard Template Library (STL)
C++ STL was extensively used to implement:

- **Dynamic arrays using vector**
- **Stacks and queues using built-in containers**
- **Algorithms such as sort(), find(), and accumulate()**

---

7. Command Line / Terminal
The terminal was occasionally used to compile (g++) and run C++ programs manually for testing command-line-based applications and batch compilation scripts.

These tools collectively supported a structured approach to learning and implementing real-world programming concepts in C++ during the internship.

4.2 METHODOLOGY

The internship project was implemented using the C++ programming language to build foundational components of software systems, focusing on object-oriented programming (OOP), data structures, file handling, and simple simulations. The methodology followed a structured programming and modular development approach, aimed at solving real-world problems with clean, maintainable code.

---

1. Problem Definition and Requirement Analysis
Two practical problems were identified for implementation:

- **Loan Approval Prediction Simulation: Based on inputs like income, credit score, and employment status.**
- **Diabetes Detection Simulation: Based on health attributes such as glucose level, BMI, and age.**

The logic for both simulations was hard-coded using conditional statements and control structures, with no external machine learning libraries involved.

---

2. Class Design and Object-Oriented Approach

- **Classes were created to encapsulate data and functionality for each system.**
- **For example, a class LoanApplicant might hold applicant details and implement a method isEligible() to simulate loan approval logic.**
- **Similarly, a class Patient could store health metrics and implement checkDiabetes() based on preset thresholds.**

This approach ensured reusability, maintainability, and clear separation of concerns.

---

3. Data Input and Validation

- **Input was taken from the user via console using standard C++ I/O (cin, cout).**
- **Validation logic was included to ensure data correctness (e.g., checking if age or income was non-negative).**
- **In some cases, static arrays or structures were used to simulate datasets.**

---

4. Decision Logic Implementation

- **For diabetes detection: simple rules were applied (e.g., if glucose > 140 and BMI > 30, predict diabetic).**
- **For loan approval: conditions like income threshold, credit history, and employment status were used to simulate approval/rejection.**
- **No machine learning algorithms were used, but the structure simulated real-world decision-making logic.**

---

5. File Handling (Optional Enhancement)

- **In advanced versions, C++ file streams (fstream) were used to:**
  - **Save applicant or patient data to a file**
  - **Read back data for future reference or review**
- **This demonstrated practical file input/output handling in C++.**

---

6. Output and Evaluation

- **Results (approval status or diabetes prediction) were displayed directly to the console.**
- **Sample test cases were used to validate correctness of logic under different input conditions.**
- **Emphasis was placed on clean formatting and informative messages to improve user interaction.**

This methodology allowed the application of core C++ concepts in a practical and structured way, consistent with the certificate provided and the internship learning objectives.

CHAPTER 5: RESULTS AND DISCUSSION

This chapter presents the results and key observations derived from the implementation of a Browser History Management System using C++. The system simulates a simplified browser experience where users can visit websites, view browsing history, and delete specific or entire browsing records via a console-based interface.

5.1 FUNCTIONAL VALIDATION

The application was tested against its expected functionalities:

- **Adding URLs to History: Websites entered by the user were successfully stored in a dynamic data structure (Linked List or Stack).**
- **Viewing History: The system could display all visited URLs in the correct order.**
- **Deleting History: Users could delete individual URLs or clear the entire history, with proper memory deallocation.**
- **Exit Operation: The system exited cleanly without memory**

leaks, confirming successful resource management.

5.2 OUTPUT SNAPSHOTS (AS PER CERTIFICATE)

As demonstrated in the certificate screenshots:

- **The program correctly accepted multiple URL entries.**
- **Displayed browsing history in reverse chronological order (most recent first).**
- **Successfully removed a specific URL from the middle of the history.**
- **Cleared all entries when prompted and exited gracefully.**

These results validated the functionality described in the certificate submitted during the internship evaluation.

5.3 OBSERVATIONS

- **The stack-based structure ensured that the most recently visited websites appeared first in the history, replicating actual browser behavior.**
- **Dynamic memory allocation allowed the system to handle an arbitrary number of entries without predefined limits.**
- **Edge cases, such as deleting from an empty history or attempting to delete a non-existent URL, were handled with appropriate user messages.**
- **The menu-driven interface provided a simple and intuitive way for users to interact with the system.**

5.4 CHALLENGES FACED

- **Memory Management: Managing dynamic memory (using new and delete) required careful implementation to prevent memory leaks.**
- **Error Handling: Ensuring the program handled invalid inputs gracefully (e.g., deleting a non-existent URL).**
- **Linked List Bugs: Edge cases such as deleting the head or tail node initially caused pointer issues during testing.**

5.5 SOLUTIONS IMPLEMENTED

- **Proper Deallocation: Used destructors and explicit delete operations to clean up memory when URLs were removed or the program exited.**
- **Modular Design: Functions like addURL(), displayHistory(), and deleteURL() were written modularly for better readability and debugging.**
- **Validation Checks: Conditions were added to check for empty history and invalid operations before proceeding, preventing runtime errors.**
- **Console Interface Enhancements: Improved menu and user prompts for smoother interaction during runtime.**

5.6 FUTURE IMPROVEMENTS

- **Persistent Storage: Add file I/O so browsing history is saved and restored between sessions.**
- **Search Functionality: Allow users to search for a specific website in the history.**
- **GUI Integration: Replace the console interface with a graphical user interface for a more user-friendly experience.**
- **Timestamping: Record the date and time of each visited URL for improved tracking.**

CHAPTER 6 : CONCLUSION

1.1 SUMMARY

The successful completion of our summer internship project titled "Browser History Management System using C++" marks an important milestone in our journey as aspiring software developers. This project provided us the opportunity to apply core programming concepts to build a functional and interactive system that simulates how web browsers manage and track user history.

The main objective was to design a console-based browser history manager that allows users to:

- **Visit (add) websites,**
- **View their current browsing history, and**
- **Delete specific websites or clear all history.**

Our system used data structures like linked lists or stacks to maintain browsing records in a dynamic and efficient way. The project incorporated critical concepts such as:

- **Dynamic memory allocation and deallocation using new and delete,**
- **Menu-driven interaction for enhanced usability,**
- **Input validation and safe pointer handling to avoid runtime errors.**

Throughout the development process, we addressed key aspects of system design:

- **Efficient handling of history data in reverse chronological order,**
- **Modular function design for readability and maintainability,**
- **Prevention of memory leaks through proper resource management.**

We encountered and resolved several challenges:

- **Managing pointer logic while deleting specific nodes (especially the first or last),**
- **Handling operations on an empty history list gracefully,**
- **Ensuring no memory leaks by tracking all dynamically allocated memory.**

This project strengthened our technical capabilities in:

- **Using object-oriented programming principles in C++,**
- **Implementing real-time user interaction via console interfaces,**
- **Applying data structure knowledge to solve practical problems,**
- **Practicing safe memory management in non-garbage-collected environments.**

Beyond technical learning, the internship also helped us build soft skills, including:

- **Communication through documentation and peer review,**
- **Time management during the iterative development cycle,**
- **Collaboration when debugging and testing across environments.**

Looking ahead, the system can be extended with features such as:

- **Persistent storage using file handling to maintain history across sessions,**
- **Timestamping and categorization of URLs,**
- **GUI-based interaction using frameworks like Qt or web-based dashboards,**
- **Integration with browser extensions or real-time web APIs.**

In conclusion, this internship not only fulfilled our academic objectives but also offered us real-world exposure to the software development lifecycle. It reinforced our confidence in building practical systems that can be further developed into production-level applications. This experience has inspired us to keep exploring impactful software solutions in the domain of user-centric tools.

1.2 FUTURE SCOPE

While the current version of the Browser History Management System effectively handles core functionalities such as adding, viewing, and deleting browser history entries, there is significant potential for enhancement and real-world application. Future improvements can focus on performance, usability, and extensibility:

• Persistent Storage

Incorporate file handling to store browser history in a local file, allowing users to retain their browsing data across multiple sessions. This could involve saving entries to a .txt or .csv file and loading them on program start.

• Timestamp Integration

Include date and time of each visited website using the C++ <ctime> library. This will help users track their browsing activity in chronological order and enable

sorting/filtering by time.

• GUI-Based Interface

Develop a graphical user interface (GUI) using libraries such as Qt, SFML, or integrate with web technologies to replace the current command-line interface. A GUI will make the system more intuitive and user-friendly for non-technical users.

• Advanced Operations

Add additional features such as:

- **Search by keyword or domain within history.**
- **History categorization based on visit frequency or content type (e.g., educational, social).**
- **Undo last deletion using stacks for recovery.**
- **Session-wise segregation for better user tracking.**

• Multi-User Support

Introduce login-based access allowing different users to manage separate browser history records. This would simulate how real browsers store history profiles for different users.

• Performance Optimization

Optimize data structure usage by exploring alternatives such as doubly linked lists or balanced trees to speed up history traversal, especially when handling large data volumes.

• Security Features

Implement password protection for clearing or modifying history to prevent unauthorized access or tampering with data. Encryption of stored history files could also be explored for enhanced privacy.

• Mobile/Desktop Application Packaging

Package the application as a lightweight desktop executable or cross-platform mobile app (using tools like C++ with Flutter or Qt for mobile) to extend usability beyond a coding environment.

• Cloud Integration

Enable cloud-based syncing of browser history using web APIs or RESTful services, allowing history access from different devices under the same user account.

CHAPTER 7: REFERENCES

 Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley. https://www.stroustrup.com/4th.html
 Schildt, H. (2017). *C++: The Complete Reference* (4th ed.). McGraw-Hill Education.
 GeeksforGeeks. (n.d.). *Linked List in C++*. https://www.geeksforgeeks.org/data-structures/linked-list/
 cplusplus.com. (n.d.). *C++ Standard Library Reference Documentation*. https://cplusplus.com/reference/
 Stack Overflow. (n.d.). Discussions and code solutions related to C++ file handling and linked list operations. https://stackoverflow.com/
 TutorialsPoint. (n.d.). *C++ File Handling*. https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm
 W3Schools. (n.d.). *C++ Basics, Functions, and OOP Concepts*. https://www.w3schools.com/cpp/
 Programiz. (n.d.). *C++ Data Structures and File I/O*. https://www.programiz.com/cpp-programming