**ADDRESS HANDBOOK**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology/Master of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

Submitted by

A.Naga Visalakshmi – AP22110010917

L.Manvitha – AP22110010939

K.Meghana – AP22110010968

V.Manasa – AP22110010977

**A picture containing text

Description automatically generated**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[December, 2023]**

# Certificate

Date: 08-Dec-23

This is to certify that the work present in this Project entitled “**ADDRESS HANDBOOK**” has been carried out by **VISHALI, MANVITHA, MANASA, MEGHANA** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in the **School of Engineering and Sciences**.

**Supervisor**

(Signature)

Prof.Ms.Kavita

Designation,

Affiliation.

# Acknowledgments:

# The provided C++ code is an implementation of a simple address book program. It includes functions to add new addresses to a vector-based address book and display the existing addresses. The program runs in a loop, allowing the user to choose between adding the user to choose between adding addresses, displaying addresses, or exiting the program.

# 

# 

# Table of Contents

certificate……………………………………………………………………….1

Acknowledgement…………………………………………………………..2

Table of contents…………………………………………………………….3

Abstract………………………………………………………………………4

List of Figures……………………………………………………………….

1. Introduction……………………………………………………………5

2.Purpose………………………………………………………………….6

3. Scope…………………………………………………………………….6

4. Discussion……………………………………………………………….7

5.Concluded Remark……………………………………………………..16

6.FutureWorks……………………………………………………………..17

7.Reference………………………………………………………………….18

# Abstract

The Address Handbook is a ground-breaking manual created to help people make sense of the complicated world of addresses. Understanding and managing addresses has grown more crucial due to the necessity for effective postal services, the growth of digital communication, and the rapid expansion of cities. The goal of this manual is to give readers a thorough understanding of addresses, including their historical significance, formatting guidelines, and useful applications in a variety of situations.

The first section of the handbook takes readers on a tour through the historical development of addresses, from ancient civilizations to the present. It explores the role that addresses play in trade, communication, and the formation of communities. Readers can better appreciate how addresses have formed the world we live in now by knowing the historical background.

The manual then goes into detail on what constitutes a complete address, dissecting the constituent parts. Readers will acquire a comprehensive understanding of how to interpret and structure addresses correctly, covering everything from street names and numbers to postal codes and nation designations. To help with understanding, there are pictures and real-world instances given.

The manual also offers insights into the function of addresses in several industries, including emergency services, e-commerce, and postal services. The usage of addresses in correct emergency responses, package tracking, and mail delivery will be revealed to readers. The handbook also examines the innovations and difficulties in addressing systems globally, emphasizing the significance of adjusting to societal shifts and technological breakthroughs.

the handbook's usefulness, a section devoted to addressing management is provided. To guarantee dependable deliveries and smooth communication, readers will acquire practical tips for maintaining and updating their address books, whether they are personal or professional. There are also pointers on how to use address verification tools, maintain digital address databases, and stay on top of address changes.

To sum up, the Address Handbook is a priceless tool for anyone attempting to understand the complex world of addresses. This thorough book gives readers the information and resources they need to comprehend, manage, and use addresses efficiently in a world that is becoming more and more connected, whether they are using them for personal or professional purposes.

**1.Introduction :**

A book or database used to hold records known as contacts is termed an address book, name and address book, or simply an address book. A few basic fields (such as first and last names, firm names, addresses, phone numbers, email addresses, fax numbers, and mobile phone numbers) are typically included in each contact entry. Most of these systems arrange the information according to the names of the individuals; but, with paper-based address books, entries may become jumbled as more people's information is added or as people relocate. Numerous address books employ tiny ring binders that let you add, remove, and rearrange pages to create more space.

In the dynamic world of today, accurate and up-to-date contact information is crucial for effective communication and collaboration. An address handbook serves as a comprehensive guide to connecting with individuals and organizations, streamlining communication channels, and fostering a well-connected network. This project aims to develop a comprehensive address handbook that will be a valuable resource for our organization.

A basic console application that lets a user manage an address book is written in C++. The following features are offered by the program. It specifies an address structure with fields for the name, street, city, state, and ZIP code of an individual. This structure represents an individual's address. Two functions are defined in the code: add Address: To add a new address to the address book, use this function. When a new address is entered, the user is prompted, and the information is stored in the address book vector. Display Addresses: All of the addresses that are presently kept in the address Book vector are shown by this function. In the primary role: To hold addresses, it first initializes an empty vector called Address Book. It then starts a while loop that lasts until the user decides to end it. A prompt asks the user to enter their selection. Depending on the option selected by the user, the application either exits if option 3 is chosen or runs the associated function, such as display Addresses to display the addresses or add Address to add an address. The user can add more addresses and have them shown while the software keeps running in a loop until they decide to stop by choosing option 3. This code is a basic illustration of how to use standard Input/output and C++ to create and maintain a basic address book. In addition to adding addresses, users can browse the stored addresses and stop the program**.**

**Purpose:**

The goal of this project is to compile an extensive address book with current, accurate contact details for all staff members, departments, and outside partners. All authorized personnel will have access to this manual, which will also act as a central database for contact details.

**Scope:**

The scope of this project encompasses the compilation, organization, and presentation of contact information for all relevant individuals and entities. This includes :

\* Employee contact information, including names, email addresses, phone numbers, and office locations

\* Departmental contact information, including department names, email addresses, phone numbers, and mailing addresses

\* External partner contact information, including company names, contact persons, email addresses, phone numbers, and mailing addres

**2. Methodology:**

The following methodology will be employed to develop the address handbook:

1. Data Collection: Gather contact information from various sources, including existing records, employee directories, and external partner databases.

2. Data Verification: Verify the accuracy and completeness of the collected data by cross-referencing with multiple sources and contacting individuals directly if necessary.

3. Data Organization: Organize the contact information in a user-friendly and searchable format, utilizing appropriate headings, subheadings, and formatting styles.

4. Data Presentation: Present the contact information in a clear, concise, and visually appealing manner, ensuring easy navigation and accessibility.

Name, street, city, state, and ZIP code fields are part of the address struct, which specifies the data type for each address.

The address book of std::vector With flexible addition and deletion, all of the addresses in the book are dynamically stored in this vector.

Features:

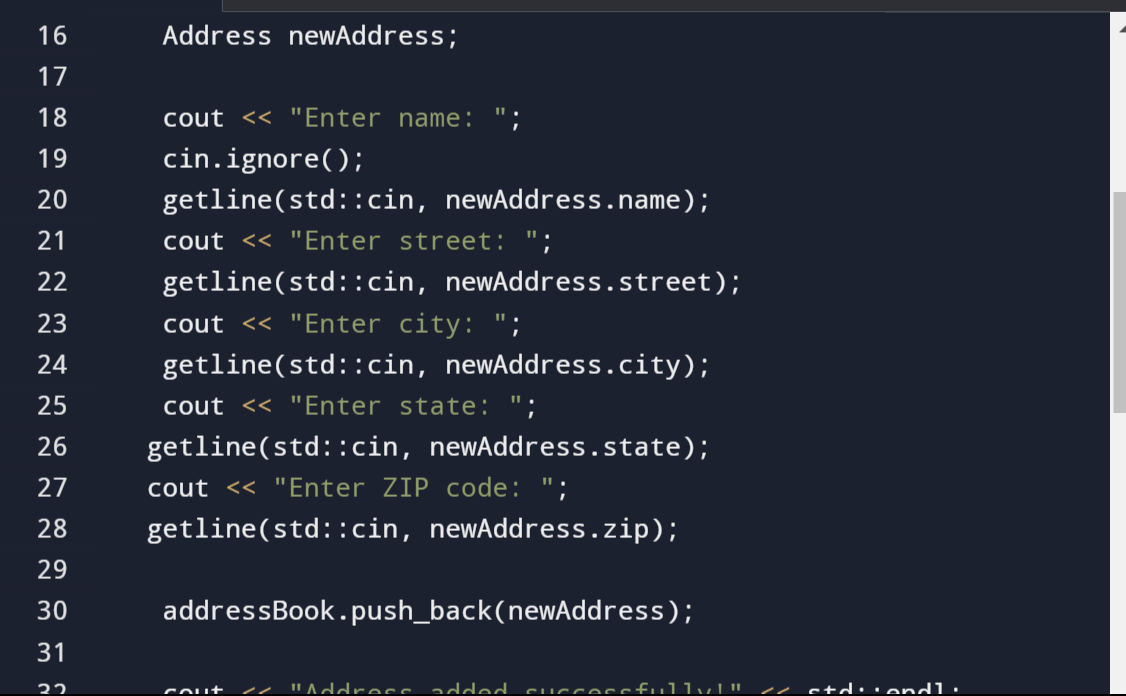
addAddress: This function adds a new Address object to the addressBook vector by asking the user for each address field, then constructs the object using the information provided. Additionally, it offers congratulations messages for a job well done.

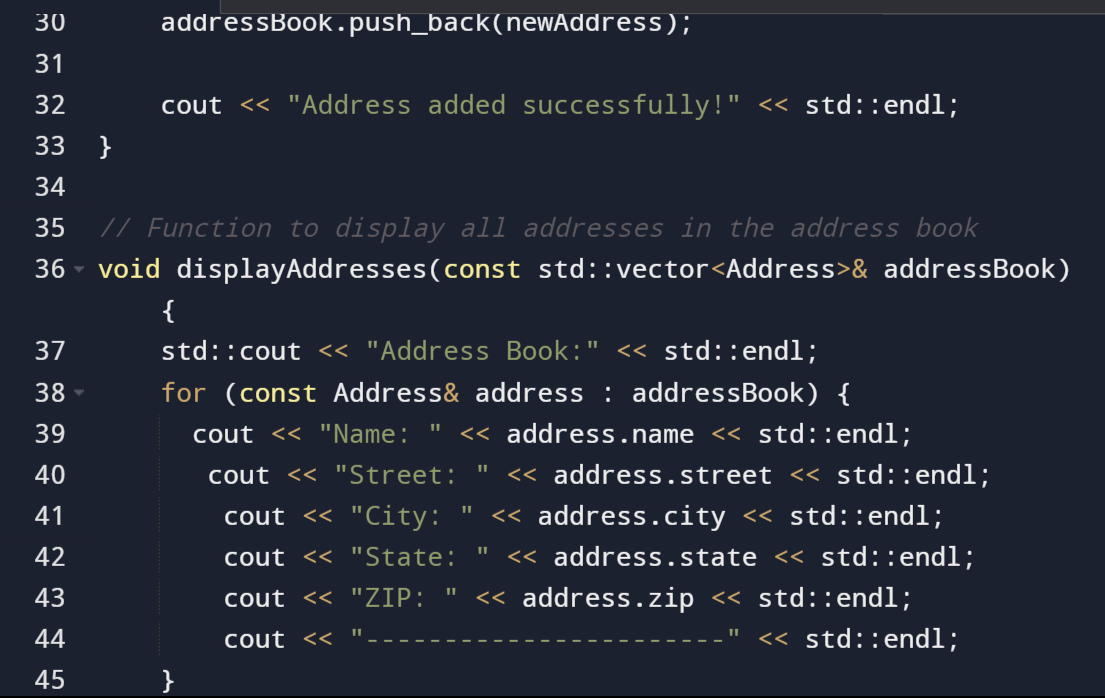
displayAddresses: Iterating through every address in the addressBook vector, this function outputs each field—name, street, etc.—along with a separating line to ensure clarity.

Scheme Structure:

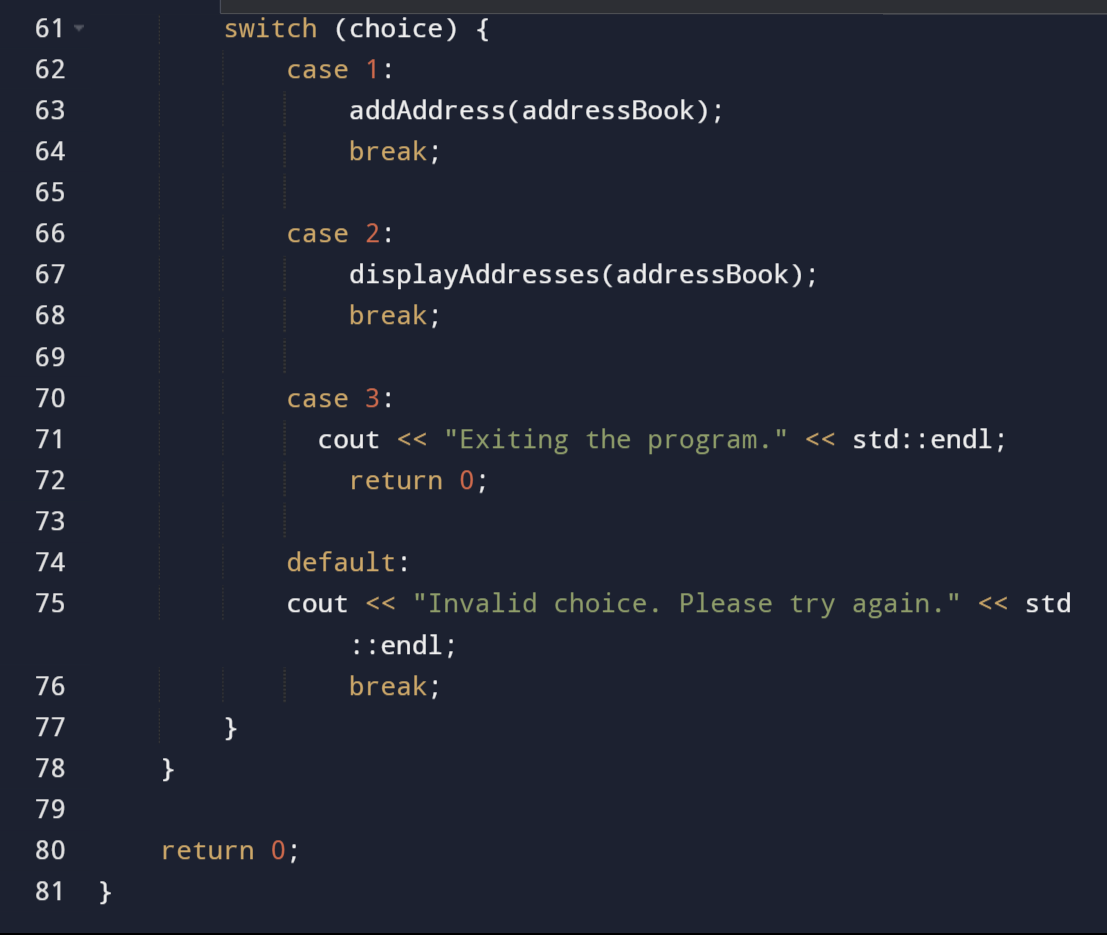
Main loop: A menu containing options to add, display addresses, and exit is displayed before the program starts an indefinite loop.

CODE SCREENSHOTS









# DISCUSSION:

This C++ code defines a simple address book program that allows users to add addresses and display the existing addresses. Let's go through the code and discuss its functionality and potential improvements:

Address Structure:

struct Address {

std::string name;

std::string street;

std::string city;

std::string state;

std::string zip;

};

The Address structure represents an individual address with fields for name, street, city, state, and ZIP code.

Function to Add an Address:

void addAddress(std::vector<Address>& addressBook) {

}

This function prompts the user to input address details and adds the new address to the vector addressBook.

Function to Display Addresses

void displayAddresses(const std::vector<Address>& addressBook) {

}

This function iterates through the addressBook vector and displays the details of each address.

Main Function:

int main() {

std::vector<Address> addressbook;

}

The main function is the entry point of the program. It contains a loop that presents a menu to the user, allowing them to add addresses, display addresses, or exit the program.Menu in Main Function:

while (true) {

}

The program continuously prompts the user for input until they choose to exit.

Switch Statement:

switch (choice) {

}

The switch statement handles the user's menu choice and calls the appropriate function.

Potential Improvement:

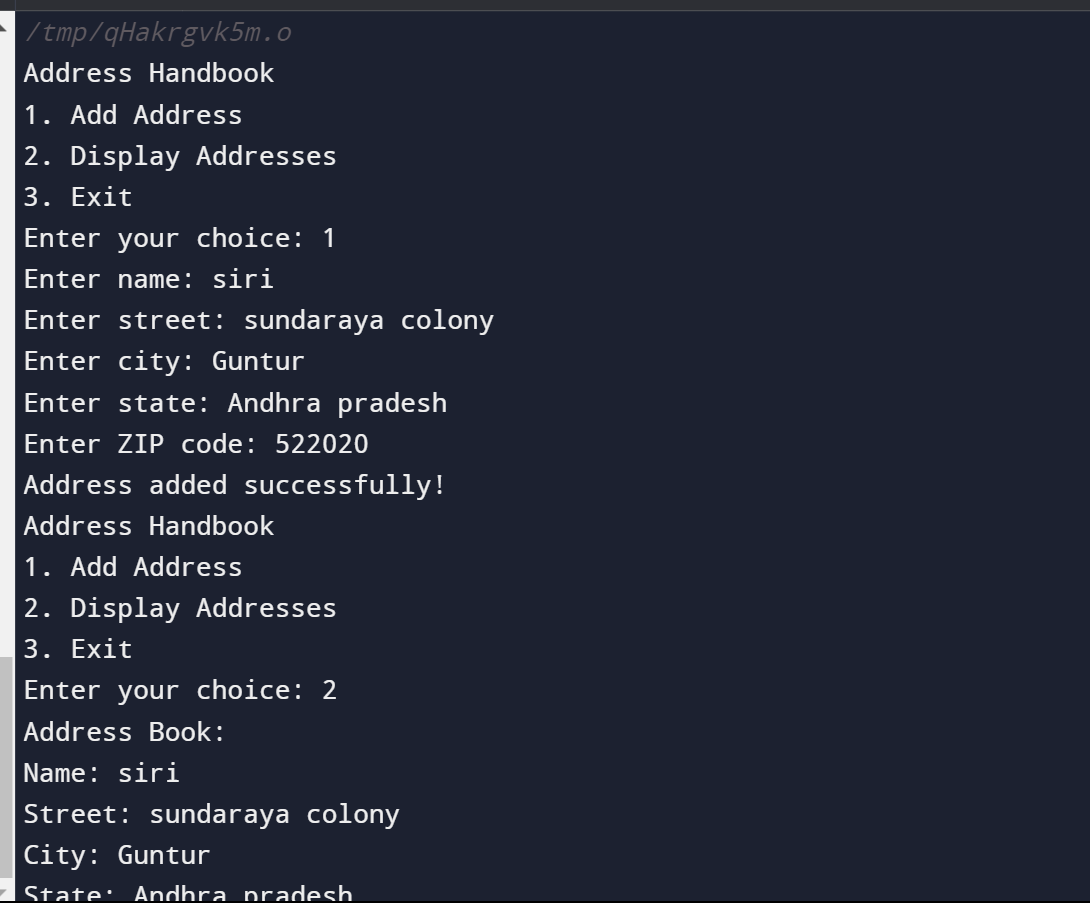
The code uses std::getline to get input for strings. However, there is a std::cin.ignore() before getting the name. This is likely used to clear the newline character left in the input buffer after a previous input operation. It would be cleaner to use std::cin.ignore() after each std::cin to handle newline characters consistently.

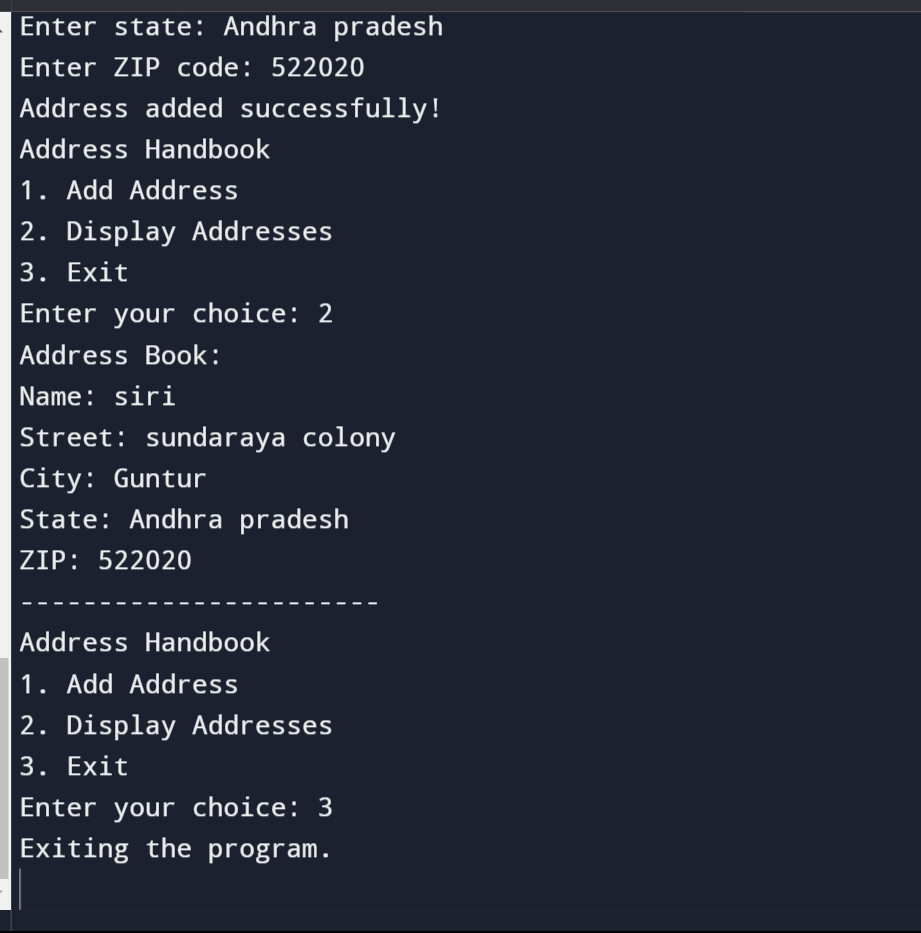
std::cout << "Enter name: ";

std::getline(std::cin, newAddress.name);

std::cin.ignore(); // Add this line to clear the newline character from the buffer

Additionally, you might want to handle invalid inputs more gracefully, especially when reading the user's choice in the menu.

OUTPUT:



Concluding Remarks:

In conclusion, the above code implements a simple address book program that allows users to add new addresses and display existing addresses. The program utilizes a struct to represent address information and a vector to store multiple address entries.

The code provides a basic menu-driven interface where users can choose between adding an address, displaying all addresses, or exiting the program. Input is obtained using std::cin and validated using std::getline. The code also handles input errors by providing a default case for invalid choices.

While the code accomplishes its intended functionality, there are opportunities for improvement and expansion. Future work could include implementing input validation, error handling, search functionality, data persistence, editing and deleting addresses, sorting addresses, and enhancing the user interface.

Overall, the code serves as a starting point for a more comprehensive and feature-rich address book application. With further enhancements, it can be tailored to meet specific requirements and provide a more robust user experience.

Future works:

Some potential future improvements and enhancements could be made to the above code. Here are a few suggestions:

1. Input validation: Currently, the code assumes that the user will always enter valid input. You could add input validation to ensure that the user enters valid data for each field, such as checking for the correct data type and length restrictions.

2. Error handling: The code does not handle any errors that may occur during execution. You could add error-handling mechanisms, such as try-catch blocks, to gracefully handle any exceptions that may occur during user input or file operations.

3. Search functionality: Adding a search function would enhance the usefulness of the address book. Users could search for specific addresses based on criteria such as name, city, or ZIP code.

4. Data persistence: Currently, the address book data is stored only in memory and is lost when the program exits. You could implement a mechanism to persist the address book data to a file, such as using file I/O operations so that the data can be saved and loaded between program runs.

Editing and deleting addresses:\* It would be helpful to provide options for editing and deleting addresses from the address book. This could be done by allowing the user to select an address from the displayed list and providing options to edit or delete it.

6. Sorting addresses: Adding the ability to sort the addresses in the address book based on different criteria, such as name or ZIP code, would make it easier for users to find and manage specific addresses.

7. User interface improvements: Consider enhancing the user interface by implementing a graphical user interface (GUI) using a library like Qt or adding color formatting and clear prompts for user input.

Reference:

<https://youtu.be/MIKWcVDPJ_0>

# 