CPP

1. CPP00
2. ex00

**spaces:**

std::cout

std::endl

In this excersize, we are using the std namespace. Namespaces help avoid naming conflicts by grouping related elements (functions, classes, variables) under a name. std is the standard namespace that contains C++'s standard library components.

**Stdio Streams**

#include <iostream>

std::cout << "\* LOUD AND UNBEARABLE FEEDBACK NOISE \*" << std::endl;

* iostream is the C++ Standard I/O library
* cout is the standard output stream (like printf in C)
* << is the stream insertion operator
* endl adds a newline and flushes the stream

1. ex01

The program requirements:

* Maximum 8 contacts
* No dynamic allocation
* Proper column formatting
* Input validation
* Command handling
* Contact field requirements

This implementation:

* A Contact class to store individual contact information
* A PhoneBook class to manage contacts
* Input validation
* Proper formatting for the SEARCH display
* Error handling
* No dynamic allocation as required

The program can:

* ADD new contacts (replaces oldest when full)
* SEARCH and display contacts in a formatted table
* EXIT the program

Let's break down the main components:

1. **Contact Class**

 Stores individual contact information

 Uses getters/setters for data access

 All data members are private for encapsulation2.

1. **PhoneBook Class**

 Manages an array of Contact objects

 Handles adding, searching, and displaying contacts

 Implements circular buffer behavior for the 8-contact limit

1. **Main Program Flow**

while (true) {

    command = getInput("Enter command (ADD, SEARCH, EXIT): ");

    if (command == "EXIT")

        break;

    else if (command == "ADD") {

        // Add new contact

    }

    else if (command == "SEARCH") {

        // Search and display contacts

    }

}

- Continuous loop for command processing

 Three main commands: ADD, SEARCH, EXIT

 Input validation and error handling

1. **Formatting and Display**

void PhoneBook::displayContactRow(int index) const {

    std::cout << "|" << std::setw(10) << index;

    // ...

}

 Fixed-width columns (10 characters)

 Right-aligned text

 Truncation with dots for long stringsWould you like me to explain any of these concepts in more detail?

**Classes and Objects**

class Contact {

private:

std::string firstName;

std::string lastName;

// ...

};

 - A class is a user-defined type that encapsulates data and functions

 Objects are instances of a class (like Contact newContact;)

 We have two classes: Contact and PhoneBook

**Access Specifiers (Visibility)**

private: // Only accessible within the class

std::string firstName;

public: // Accessible from outside the class

void setContact(...);

  private: Members can only be accessed within the class

 public: Members can be accessed from anywhere

**Member Functions**

void Contact::setContact(std::string firstName, ...) {

this->firstName = firstName;

// ...

}

 Functions that belong to a class

 Can access private members of the class

 The this pointer refers to the current object

**Const Member Functions**

std::string getFirstName() const { return firstName; }

 - The const keyword promises the function won't modify class members

 Important for read-only operations

**Constructors**

Contact::Contact() {}

PhoneBook::PhoneBook() : currentSize(0), oldestIndex(0) {}

- Special member functions that initialize objects

 The PhoneBook constructor uses an initialization list

**Arrays and Static Memory**

Contact contacts[8]; // Fixed-size array of Contact objects

 No dynamic allocation (no new or delete)

 Fixed size of 8 contacts as required

**References and Parameters**

void addContact(const Contact& contact)

  const prevents modification of the parameter

 & passes by reference to avoid copying

**String Handling**

std::string firstName;

std::string truncate(std::string str) const

 sing C++ string class instead of C-style strings

 String manipulation methods like substr()

**Input/Output Streams**

std::cout << "Enter command (ADD, SEARCH, EXIT): ";

std::getline(std::cin, input);

  iostream for input/output operations

 iomanip for formatting output (setw)

**Exception Handling**

try {

int index = std::stoi(indexStr);

// ...

} catch (...) {

std::cout << "Invalid index!" << std::endl;

}

try {

int index = std::stoi(indexStr);

// ...

} catch (...) {

std::cout << "Invalid index!" << std::endl;

}

 - Using try-catch blocks for error handling

 Catching invalid input conversions