**Experiment 1**

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)**

|  |  |
| --- | --- |
| Roll No. C013 | Name: Ashmit Jain |
| Program : BTI | Division: B1 |
| Batch: 1 | Date of Experiment: |
| Date of Submission: 8/6/24 | Grade : |

* 1. **Tasks given in PART A to be completed here**

*(****Students must write the answers of the task(s) given in the PART A )***

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class Library {**

**private:**

**string bookName;**

**float bookCost;**

**bool issuedStatus;**

**int serialNumber;**

**public:**

**// Default constructor to initialize values**

**Library() : bookName(""), bookCost(0.0), issuedStatus(false), serialNumber(0) {}**

**// Parameterized constructor**

**Library(string name, float cost, bool status, int serial)**

**: bookName(name), bookCost(cost), issuedStatus(status), serialNumber(serial) {}**

**// Function to input book details**

**void inputBookDetails() {**

**cout << "Enter book name: ";**

**cin.ignore();**

**getline(cin, bookName);**

**cout << "Enter book cost: ";**

**cin >> bookCost;**

**cout << "Is the book issued? (1 for Yes, 0 for No): ";**

**cin >> issuedStatus;**

**cout << "Enter serial number: ";**

**cin >> serialNumber;**

**}**

**// Function to display book details**

**void displayBookDetails() const {**

**cout << "Book Name: " << bookName << endl;**

**cout << "Book Cost: " << bookCost << endl;**

**cout << "Issued Status: " << (issuedStatus ? "Issued" : "Not Issued") << endl;**

**cout << "Serial Number: " << serialNumber << endl;**

**}**

**// Function to display all books**

**static void displayAllBooks(Library books[], int size) {**

**for (int i = 0; i < size; ++i) {**

**cout << "\nBook " << i + 1 << " details:\n";**

**books[i].displayBookDetails();**

**}**

**}**

**// Function to display a specific book by serial number**

**static void displaySpecificBook(Library books[], int size) {**

**int serial;**

**cout << "Enter serial number: ";**

**cin >> serial;**

**for (int i = 0; i < size; ++i) {**

**if (books[i].serialNumber == serial) {**

**books[i].displayBookDetails();**

**return;**

**}**

**}**

**cout << "Book not found.\n";**

**}**

**// Function to insert a book at a specified position**

**static void insertAtPosition(Library books[], int &size, int pos, Library newBook) {**

**for (int i = size; i > pos; --i) {**

**books[i] = books[i - 1];**

**}**

**books[pos] = newBook;**

**++size;**

**}**

**// Function to insert a book at the start**

**static void insertAtStart(Library books[], int &size, Library newBook) {**

**insertAtPosition(books, size, 0, newBook);**

**}**

**// Function to insert a book at the end**

**static void insertAtEnd(Library books[], int &size, Library newBook) {**

**books[size] = newBook;**

**++size;**

**}**

**// Function to delete a book at the start**

**static void deleteAtStart(Library books[], int &size) {**

**for (int i = 0; i < size - 1; ++i) {**

**books[i] = books[i + 1];**

**}**

**--size;**

**}**

**// Function to delete a book at the end**

**static void deleteAtEnd(Library books[], int &size) {**

**--size;**

**}**

**// Function to delete a book at a specified position**

**static void deleteAtPosition(Library books[], int &size, int pos) {**

**for (int i = pos; i < size - 1; ++i) {**

**books[i] = books[i + 1];**

**}**

**--size;**

**}**

**};**

**int main() {**

**int n, choice;**

**cout << "Enter the number of books: ";**

**cin >> n;**

**// Create an array of Book objects**

**Library books[100]; // Assuming a maximum of 100 books**

**int size = n;**

**// Input details for each book**

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

**cout << "\nEntering details for book number " << i + 1 << ":\n";**

**books[i].inputBookDetails();**

**}**

**do {**

**cout << "\nEnter your choice:\n";**

**cout << "1. Display all the books\n";**

**cout << "2. Display specific book\n";**

**cout << "3. Insert at position\n";**

**cout << "4. Insert at start\n";**

**cout << "5. Insert at end\n";**

**cout << "6. Delete at start\n";**

**cout << "7. Delete at end\n";**

**cout << "8. Delete at position\n";**

**cout << "9. Exit\n";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**Library::displayAllBooks(books, size);**

**break;**

**case 2:**

**Library::displaySpecificBook(books, size);**

**break;**

**case 3: {**

**Library newBook;**

**newBook.inputBookDetails();**

**int pos;**

**cout << "Enter position to insert: ";**

**cin >> pos;**

**Library::insertAtPosition(books, size, pos - 1, newBook);**

**break;**

**}**

**case 4: {**

**Library newBook;**

**newBook.inputBookDetails();**

**Library::insertAtStart(books, size, newBook);**

**break;**

**}**

**case 5: {**

**Library newBook;**

**newBook.inputBookDetails();**

**Library::insertAtEnd(books, size, newBook);**

**break;**

**}**

**case 6:**

**Library::deleteAtStart(books, size);**

**break;**

**case 7:**

**Library::deleteAtEnd(books, size);**

**break;**

**case 8: {**

**int pos;**

**cout << "Enter position to delete: ";**

**cin >> pos;**

**Library::deleteAtPosition(books, size, pos - 1);**

**break;**

**}**

**case 9:**

**cout << "Exiting...\n";**

**break;**

**default:**

**cout << "Invalid Input\n";**

**break;**

**}**

**} while (choice != 9);**

**return 0;**

**}**

**Observations and Learning:**

*(****Students must write the observations and learning based on their understanding built about the subject matter and inferences drawn)***

**Class Definition:**

The Library class contains private members: bookName, bookCost, issuedStatus, and serialNumber.

It has a default constructor and a parameterized constructor for initialization.

Public methods include functions for inputting and displaying book details, and various static methods for handling operations on an array of Library objects.

**Book Management:**

The inputBookDetails function gathers details of a book from the user.

The displayBookDetails function prints the details of a book.

displayAllBooks and displaySpecificBook functions allow for viewing all books or a specific book based on serial number.

**Array Operations:**

Static methods insertAtPosition, insertAtStart, insertAtEnd, deleteAtStart, deleteAtEnd, and deleteAtPosition manage the array of books, allowing insertion and deletion of books at specified positions.

**Main Function:**

The main function initializes an array of Library objects and collects book details from the user.

A menu-driven interface allows the user to choose various operations on the books array, including displaying all books, displaying a specific book, inserting books at various positions, and deleting books from various positions.

**Menu and Input Handling:**

The program uses a do-while loop to repeatedly present a menu to the user until they choose to exit.

The user inputs their choice, and the program executes the corresponding operation.

###### Conclusion:

(Students must write the conclusive statements as per the attainment of individual outcomes listed above and learning/observation noted in section B.2)

This program effectively demonstrates how to manage a collection of books using an object-oriented approach in C++. By encapsulating book properties and operations within the Library class, the program maintains a clean and modular structure. Static methods facilitate operations on an array of Library objects, showcasing how class functions can be used to manipulate class instances collectively. The menu-driven interface allows users to interact with the program intuitively, performing various operations such as insertion, deletion, and display of books. Overall, the program provides a practical example of object-oriented programming concepts, including encapsulation, constructors, and static methods, applied to a real-world scenario of library management.

###### Question of curiosity:

###### What is the difference between int array[] and int[] array?

###### C++ Array Declarations

###### Int array[]:

###### Usage: This form is used when the array is being declared and its size is either inferred from initialization or not explicitly specified.

###### Example: int array[] = {1, 2, 3, 4, 5}; // The size is inferred to be 5 based on the number of initializers

###### Notes:

###### \\When initializing an array this way, the compiler determines the size of the array based on the number of elements in the initializer list.

###### Int array[5]:

###### Usage: This form explicitly specifies the size of the array.

###### Example:

###### int array[5]; // Declares an array of 5 integers

###### array[0] = 10; // Assigning a value to the first element

###### Key Points:

###### Initialization and Size:

###### int array[] can be used with an initializer list where the size is automatically determined by the number of elements provided.

###### int array[5] explicitly defines the size of the array, and you must ensure that you do not exceed this size when accessing or modifying the array.

###### Flexibility:

###### int array[] is useful when you want the compiler to automatically handle the size of the array based on the number of initial values provided.

###### int array[5] is used when you know the exact size of the array in advance and do not need initialization or want to define the size explicitly without initial values.

###### Can we declare array size as negative? Justify

###### No, in C++, you cannot declare an array with a negative size. The reasons are both logical and technical:

###### Logical Inconsistency:

###### An array size represents the number of elements it can hold.

###### A negative number of elements is nonsensical as you cannot have a collection with fewer than zero elements.

###### Memory Allocation:

###### Arrays require a specific amount of memory to be allocated at compile time.

###### A negative size would lead to undefined behavior because the memory allocation routines expect a non-negative size.

###### Compiler Enforcement:

###### Most C++ compilers will flag the use of a negative array size as a compile-time error.

###### For example: int arr[-5]; // This will cause a compile-time error

###### 

###### Standard Compliance:

###### The C++ Standard explicitly defines array sizes to be of type `size\_t`, which is an unsigned integral type. This inherently prevents negative sizes since `size\_t` can only represent non-negative values.

###### Trying to declare an array with a negative size will result in a compile-time error similar to: error: size of array ‘arr’ is negative

###### Therefore, it is both logically invalid and technically unsupported to declare an array with a negative size in C++.

###### What are the advantages and disadvantages of an array?

###### Advantages of Arrays

###### Fixed Size: The size of an array is determined at the time of its creation and cannot be changed. This makes memory management straightforward.

###### Efficient Access: Arrays provide O(1) time complexity for accessing elements by index, making them very efficient for lookups.

###### Memory Contiguity: Elements in an array are stored in contiguous memory locations, which can lead to improved cache performance and faster access times.

###### Ease of Use: Arrays are simple to understand and use, with straightforward syntax for defining, accessing, and iterating over elements.

###### Static Allocation: In languages like C and C++, arrays can be statically allocated, reducing the overhead of dynamic memory allocation and deallocation.

###### Efficiency for Iteration: Iterating through an array is typically very efficient due to its contiguous memory layout.

###### Disadvantages of Arrays

###### Fixed Size: The fixed size of arrays can be a limitation if the required size is not known at compile time or if the array needs to grow or shrink dynamically.

###### Wasted Space: If the array size is overestimated, there can be wasted memory space. Conversely, underestimating the size requires resizing, which is not straightforward for statically allocated arrays.

###### Insertion and Deletion: Inserting or deleting elements from an array (except at the end) can be inefficient, as it may require shifting multiple elements, leading to O(n) time complexity.

###### Lack of Flexibility: Arrays do not provide built-in mechanisms for dynamic resizing, complex data structures, or advanced operations such as sorting or searching.

###### Type Homogeneity: Arrays typically require all elements to be of the same type, which can be restrictive if different types need to be stored together (although this can be mitigated with structures or classes in languages like C++).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*