Class Templates with Multiple Parameters:

Write a class template Pair that can store a pair of values of any two data types. Include member functions to set and get the values. Demonstrate the usage of this template with different data types.

#include <iostream>

template <typename T1, typename T2>

class Pair {

private:

T1 first;

T2 second;

public:

Pair() {}

Pair(T1 f, T2 s) : first(f), second(s) {}

void setFirst(T1 f) {

first = f;

}

void setSecond(T2 s) {

second = s;

}

T1 getFirst() const {

return first;

}

T2 getSecond() const {

return second;

}

};

int main() {

Pair<int, double> p1(1, 3.14);

Pair<std::string, char> p2("Hello", 'C');

p1.setFirst(42);

p1.setSecond(2.718);

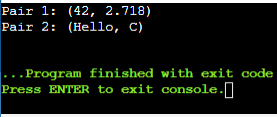
std::cout << "Pair 1: (" << p1.getFirst() << ", " << p1.getSecond() << ")\n";

std::cout << "Pair 2: (" << p2.getFirst() << ", " << p2.getSecond() << ")\n";

return 0;

}

Output:



Pointer to Objects:

Define a class Book with data members title and author. Create an array of pointers to Book objects. Write functions to input details for each book, display the details, and search for a book by title.

include <iostream>

#include <string>

class Book {

private:

std::string title;

std::string author;

public:

Book(const std::string& t, const std::string& a) : title(t), author(a) {}

void displayDetails() const {

std::cout << "Title: " << title << ", Author: " << author << std::endl;

}

const std::string& getTitle() const {

return title;

}

};

int main() {

const int MAX\_BOOKS = 3;

Book\* library[MAX\_BOOKS];

for (int i = 0; i < MAX\_BOOKS; ++i) {

std::string title, author;

std::cout << "Enter details for book " << i + 1 << ":" << std::endl;

std::cout << "Title: ";

std::getline(std::cin >> std::ws, title);

std::cout << "Author: ";

std::getline(std::cin >> std::ws, author);

library[i] = new Book(title, author);

}

std::cout << "\nLibrary Books:\n";

for (int i = 0; i < MAX\_BOOKS; ++i) {

std::cout << "Book " << i + 1 << ": ";

library[i]->displayDetails();

}

std::string searchTitle;

std::cout << "\nEnter title to search: ";

std::getline(std::cin >> std::ws, searchTitle);

bool found = false;

for (int i = 0; i < MAX\_BOOKS; ++i) {

if (library[i]->getTitle() == searchTitle) {

std::cout << "Book found at index " << i + 1 << ": ";

library[i]->displayDetails();

found = true;

break;

}

}

if (found) {

std::cout << "Book with title \"" << searchTitle << "\" not found." << std::endl;

}

for (int i = 0; i < MAX\_BOOKS; ++i) {

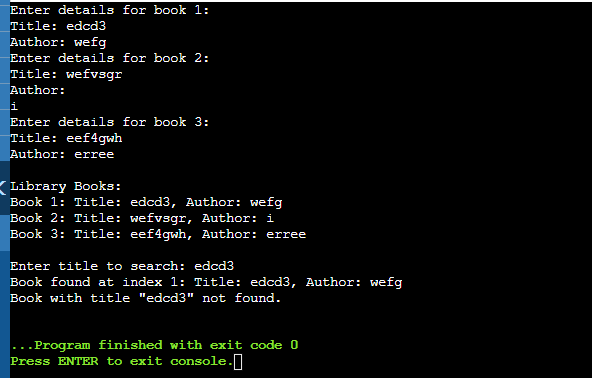
delete library[i];

}

return 0;

}

Output:



Templates:

Write a template class Array that can store an array of any data type. Include member functions to perform operations like adding an element, removing an element, and displaying the array. Demonstrate the functionality with different data types.

include <iostream>

#include <vector>

template <typename T>

class Array {

private:

std::vector<T> elements;

public:

void addElement(const T& element) {

elements.push\_back(element);

}

void removeElement(int index) {

if (index >= 0 && index < elements.size()) {

elements.erase(elements.begin() + index);

} else {

std::cout << "Invalid index for removal.\n";

}

}

void display() const {

std::cout << "Array: [";

for (size\_t i = 0; i < elements.size(); ++i) {

std::cout << elements[i];

if (i != elements.size() - 1) {

std::cout << ", ";

}

}

std::cout << "]\n";

}

};

int main() {

Array<int> intArray;

intArray.addElement(10);

intArray.addElement(20);

intArray.addElement(30);

intArray.display();

Array<double> doubleArray;

doubleArray.addElement(3.14);

doubleArray.addElement(1.618);

doubleArray.display();

Array<std::string> stringArray;

stringArray.addElement("apple");

stringArray.addElement("banana");

stringArray.addElement("cherry");

stringArray.display();

return 0;

}

Output:

