1.#include<iostream>

using namespace std;

class point{

public:

double x,y;

};

void offsetPoint(point p,double x,double y)

{

p.x += x;

p.y += y;

}

int main(){

point p;

p.x = 3.0;

p.y = 4.0;

offsetPoint(p, 1.0, 2.0);

cout <<"("<< p.x <<"," << p.y <<")";

}

2.#include<iostream>

using namespace std;

class point{

public:

double x,y;

};

void offsetPoint(point &p,int x,int y)

{

p.x += x;

p.y += y;

cout << "Inside address of object"<<&p<<endl;

}

int main(){

point p;

p.x = 3.0;

p.y = 4.0;

cout << "outside address of object"<<&p<<endl;

offsetPoint(p, 1.0, 2.0);

cout <<"x is "<<p.x<< " y is " << p.y<<endl;

cout << "outside address of object"<<&p<<endl;

}

3.#include<iostream>

using namespace std;

class point{

public:

double x,y;

point(){

x = 0.0;

y = 0.0;

cout<<"constructor called"<<endl;

}

};

void offsetPoint(point &p,int x,int y)

{

p.x += x;

p.y += y;

cout << "Inside address of object"<<&p<<endl;

}

int main(){

point p;

cout<<"after constructor called"<<endl;

p.x = 3.0;

p.y = 4.0;

cout << "outside address of object"<<&p<<endl;

offsetPoint(p, 1.0, 2.0);

cout <<"x is "<<p.x<< " y is " << p.y<<endl;

cout << "outside address of object"<<&p<<endl;

}

4.#include<iostream>

using namespace std;

class point{

public:

double x,y;

point(double p, double q){

x = p;

y = q;

cout<<" param-constructor called"<<endl;

}

};

void offsetPoint(point &p,int x,int y)

{

p.x += x;

p.y += y;

cout << "Inside address of object"<<&p<<endl;

}

int main(){

point p(13,44);

cout <<"x is "<<p.x<< " y is " << p.y<<endl;

cout<<"after constructor called"<<endl;

p.x = 3.0;

p.y = 4.0;

cout << "outside address of object"<<&p<<endl;

offsetPoint(p, 1.0, 2.0);

cout <<"x is "<<p.x<< " y is " << p.y<<endl;

cout << "outside address of object"<<&p<<endl;

}

5.#include<iostream>

using namespace std;

class Rectangle{

private:

double length;

double width;

public:

void setLength(double l)

{

length = l;

}

void setWidth(double w)

{

width = w;

}

double getLength(){

return length;

}

double getWidth(){

return width;

}

double calculateArea(){

return length \* width;

}

};

int main(){

Rectangle rect;

rect.setLength(20);

rect.setWidth(30);

cout<<"Length:"<<rect.getLength()<<endl;

cout<<"Width:"<<rect.getWidth()<<endl;

}

6.#include<iostream>

using namespace std;

class Animal{

public:

void eat(){

cout<<"Eat the animal:"<<endl;

}

};

class Dog:public Animal{

public:

void bark(){

cout<<"The dog is barking"<<endl;

}

};

int main(){

Dog myDog;

myDog.eat();

myDog.bark();

return 0;

}

**1.Default Constructor:**

**Write a class Student with a default constructor that initializes the student's name to "Unknown" and age to 0. Add a method display to print the student's details**.

#include <iostream>

using namespace std;

class Student {

private:

string name;

int age;

public:

Student() { // Default constructor

name = "Unknown";

age = 0;

}

void display() { // Method to display student's details

cout << "Name: " << name << endl;

cout << "Age: " << age << endl;

}

};

int main() {

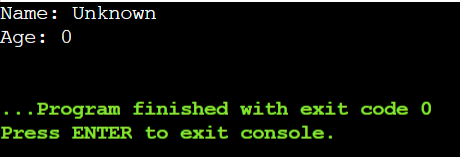
Student student; // Create a Student object using the default constructor

student.display(); // Display the student's details

return 0;

}

**Output:**



**2. \*Parameterized Constructor:\***

**Write a class Rectangle with a parameterized constructor that initializes the length and width. Add a method area that returns the area of the rectangle.**

#include <iostream>

using namespace std;

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double l, double w) {

length = l; // Initialize length with l

width = w; // Initialize length with w

}

void setLength(double l) { // Setter method for length

length = l;

}

void setWidth(double w) { // Setter method for Width

width = w;

}

double getLength() { // Getter method for length

return length;

}

double getWidth() { // Setter method for width

return width;

}

double calculateArea(){ // calculate and return the area of the rectangle

return length \* width;

}

};

int main() {

Rectangle rect(20, 30); // Create a Rectangle object named rect with length 20 and width 30 cout << "Length: " << rect.getLength() << endl;

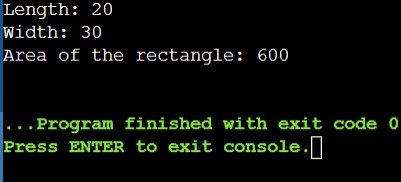
cout << "Width: " << rect.getWidth() << endl;

cout << "Area of the rectangle: " << rect.calculateArea() << endl;

return 0;

}

**Output:**



**3. \*Multiple Constructors:\***

**Write a class Book that has both a default constructor and a parameterized constructor. The default constructor should set the title to "Unknown" and the number of pages to 0. The parameterized constructor should initialize the title and pages with given values.**

#include <iostream>

#include <string>

// Use the standard library namespace to simplify code

using namespace std;

class Book {

public:

// Member variables

string title;

int pages;

// Default constructor

Book() {

title = "Unknown";

pages = 0;

}

// Parameterized constructor

Book(const char\* t, int p) {

title = t;

pages = p;

}

// Method to display book's details

void display() const {

cout << "Title: " << title << "\n";

cout << "Pages: " << pages << "\n";

}

};

int main() {

// Create a Book object using the default constructor

Book defaultBook;

defaultBook.display();

// Create a Book object using the parameterized constructor

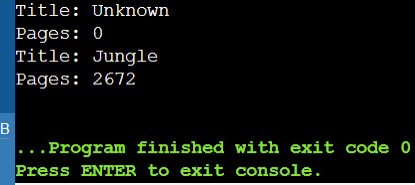
Book existingBook("Jungle", 2672);

existingBook.display();

return 0;

}

**Output:**



**4. \*Constructor Overloading:\***

**Write a class Complex that represents complex numbers. Implement a default constructor that sets both real and imaginary parts to 0, and a parameterized constructor that takes two arguments to initialize the real and imaginary parts.**

#include <iostream>

using namespace std;

class Complex {

private:

double real;

double imaginary;

public:

Complex() { // Default constructor

setReal(0);

setImaginary(0);

}

Complex(double r, double i) { // Parameterized constructor

setReal(r);

setImaginary(i);

}

void setReal(double r) {

real = r;

}

void setImaginary(double i) {

imaginary = i;

}

double getReal() {

return real;

}

double getImaginary(){

return imaginary;

}

};

int main() {

Complex defaultComplex; // Complex using the default constructor

Complex paramComplex(7.5, 8.5); // Create a Complex object using the parameterized constructor

cout << "Default Complex Number:" << endl;

cout << "Complex number: " << defaultComplex.getReal() << " + " << defaultComplex.getImaginary() << "i" << endl;

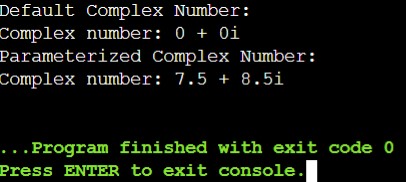
cout << "Parameterized Complex Number:" << endl;

cout << "Complex number: " << paramComplex.getReal() << " + " << paramComplex.getImaginary() << "i" << endl;

return 0;

}

**Output:**



9**. \*Pointer to an Integer:\* Write a function increment that takes a pointer to an integer and increments its value by 1. Demonstrate the function in the main program.**

#include <iostream>

using namespace std;

void increment(int\* ptr) { // Function to increment the value pointed to by ptr

(\*ptr)++; // Increment the value at the memory location pointed by ptr

}

int main() {

int num = 10;

cout << "Before increment: " << num << endl; // Print initial value of num

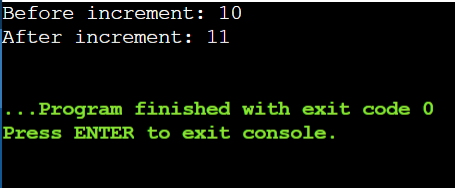
increment(&num); // Call the increment function with the address of num

cout << "After increment: " << num << endl; // Print value of num after increment

return 0;

}

**Output:**



**10. \*Pointer to a Class:\* Write a class Circle with a method area. Create a pointer to an object of this class and call the area method using the pointer.**

#include <iostream>

using namespace std;

class Circle {

private:

double radius;

public:

Circle(double r) { // Constructor

radius = r;

}

double area() {

return 3.14 \* radius \* radius; // Assuming pi is approximately 3.14

}

};

int main() {

Circle c(5.0); // Create an object of Circle

Circle\* ptrCircle = &c; // Create a pointer to the Circle object

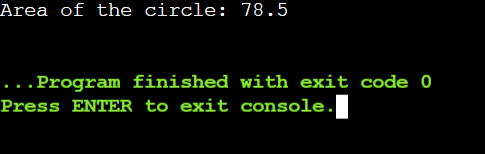
double circleArea = ptrCircle->area(); // Call the area method using the pointer

cout << "Area of the circle: " << circleArea << endl;

return 0;

}

**Output:**



11. \***Array of Pointers:\***

**Write a program that creates an array of pointers to integers. Initialize the array with values and print them using the pointers.**

#include <iostream>

using namespace std;

int main() {

int values[] = {10, 20, 30, 40, 50}; // Define an array of integers

int\* ptrArray[5]; // Array of 5 pointers

// Initialize each pointer to point to corresponding element in values array

for (int i = 0; i < 5; ++i) {

ptrArray[i] = &values[i];

}

// Print values using the pointers

cout << "Values in the array using pointers:" << endl;

for (int i = 0; i < 5; ++i) {

cout << \*ptrArray[i] << " ";

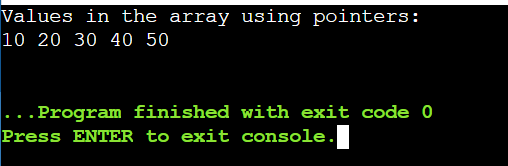
}

cout << endl;

return 0;

}

**Output:**



**12. \*Pointer to an Array:\***

**Write a function that takes a pointer to an array of integers and the size of the array. The function should print all elements of the array.**

#include <iostream>

using namespace std;

// Function to print elements of an array given a pointer and size

void printArray(int\* arr, int size) {

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

cout << arr[i] << " ";

}

cout << endl;

}

int main() {

int arr[] = {70, 80, 90, 100, 110};// Define an array of integers

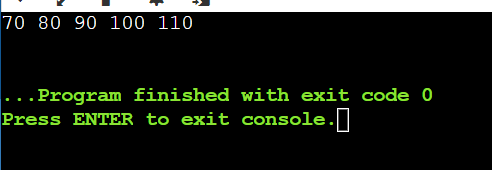
int size = sizeof(arr) / sizeof(arr[0]); // Calculate the size of the array

printArray(arr, size); // Call the function to print the array elements

return 0;

}

**Output:**



**13. \*Dynamic Memory Allocation:\***

**Write a program that dynamically allocates memory for an integer, assigns a value to it, and then frees the memory.**

#include <iostream>

using namespace std;

int main() {

int \*ptr = new int; // Dynamic memory allocation for an integer

\*ptr = 24; // Assign a value to the dynamically allocated integer

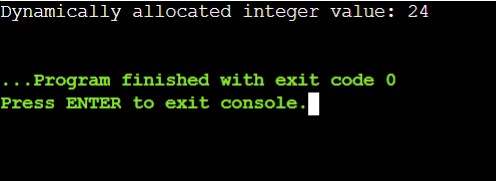
cout << "Dynamically allocated integer value: " << \*ptr << endl;

delete ptr; // Free the dynamically allocated memory

return 0;

}

**Output:**



**14. \*Reference to an Integer:\***

**Write a function swap that takes two integer references and swaps their values. Demonstrate the function in the main program.**

**#**include <iostream>

using namespace std;

void swap(int &a, int &b) {

int temp = a; // Store the value of a in a temporary variable

a = b; // Assign the value of b to a

b = temp; // Assign the stored value of a (temp) to b

}

int main() {

int num1 = 23;

int num2 = 24;

cout << "Before swap:" << endl;

cout << "num1 = " << num1 << ", num2 = " << num2 << endl;

// Call the swap function passing num1 and num2 by reference

swap(num1, num2);

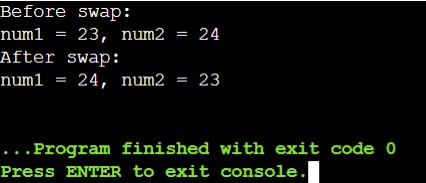
cout << "After swap:" << endl;

cout << "num1 = " << num1 << ", num2 = " << num2 << endl;

return 0;

}

**Output:**



**15. \*Reference to a Class Object:\***

**Write a class Box with a method volume. Create an object of this class and a reference to this object. Call the volume method using the reference.**

#include <iostream>

using namespace std;

class Box {

private:

double length;

double width;

double height;

public:

// Constructor to initialize dimensions

Box(double l, double w, double h) : length(l), width(w), height(h) {}

double volume() {

return length \* width \* height;

}

};

int main() {

Box myBox(6.0, 9.0, 12.0); // Create a Box object

Box &boxRef = myBox; // Create a reference to the Box object

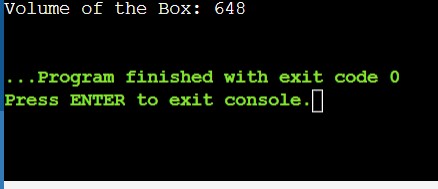
// Call the volume method using the reference

cout << "Volume of the Box: " << boxRef.volume() << endl;

return 0;

}

**Output:**



**16. \*Returning Reference from a Function:\***

**Write a function that takes an array of integers and returns a reference to the largest element. Demonstrate the function in the main program.**

#include <iostream>

using namespace std;

// Function to find the reference to the largest element in an array

int& findLargestElement(int arr[], int size) {

int maxIndex = 0;

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

if (arr[i] > arr[maxIndex]) {

maxIndex = i;

}

}

return arr[maxIndex];

}

int main() {

int numbers[] = {1,2,3,4,5,6};

// Find the reference to the largest element in the array

int &largest = findLargestElement(numbers, 6);

cout << "Largest element in the array: " << largest << endl;

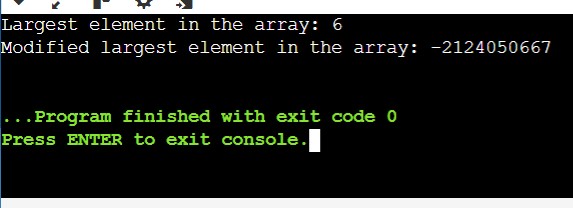
largest = 100; // Modify the largest element using the reference

cout << "Modified largest element in the array: " << numbers[findLargestElement(numbers, 6)] << endl;

return 0;

}

**Output:**



**17. \*Pass by Value:\***

**Write a function addTen that takes an integer by value and adds 10 to it. Demonstrate how the original value is not changed after calling the function.**

#include <iostream>

using namespace std;

// Function to add 10 to an integer passed by value

void addTen(int num) {

num += 10; // Add 10 to the local copy of num

cout << "Inside function - Value after adding 10: " << num << endl;

}

int main() {

int number = 7;

cout << "Original value: " << number << endl;

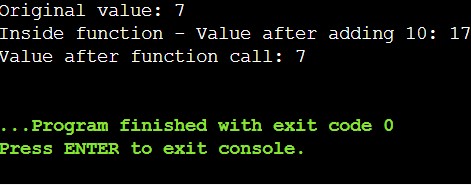
addTen(number); // Call addTen function with number passed by value

cout << "Value after function call: " << number << endl;

return 0;

}

**Output:**



**18. \*Pass by Reference:\***

**Write a function addTenRef that takes an integer by reference and adds 10 to it. Demonstrate how the original value is changed after calling the function.**

#include <iostream>

using namespace std;

// Function to add 10 to an integer passed by value

void addTenRef(int &num) {

num += 10; // Add 10 to the local copy of num

cout << "Inside function - Value after adding 10: " << num << endl;

}

int main() {

int number = 7;

cout << "Original value: " << number << endl;

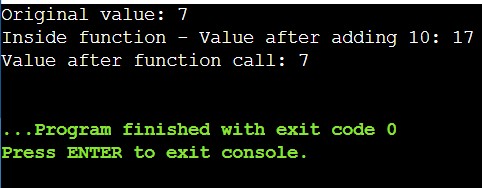
addTenRef(number); // Call addTen function with number passed by value

cout << "Value after function call: " << number << endl;

return 0;

}

**Output:**



**19. \*Function Returning a Reference:\***

**Write a function that returns a reference to a static variable. Modify the returned value in the main function and print it.**

#include <iostream>

using namespace std;

int& staticVariable() {

static int num = 10; // Static variable initialized to 10

return num; // Return reference to static variable

}

int main() {

int &ref = staticVariable(); // Get a reference to the static variable from the function

// Modify the static variable using the reference

ref = 20;

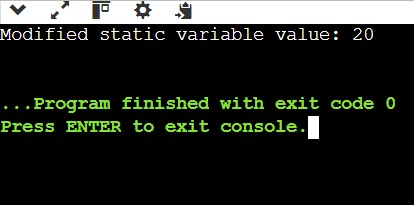
// Print the modified value of the static variable

cout << "Modified static variable value: " << staticVariable() << endl;

return 0;

}

**Output:**



**20. \*Passing Objects by Value and Reference:\***

**Write a class Employee with attributes name and salary. Write two functions: one that takes an Employee object by value and another that takes an Employee object by reference. Modify the salary in both functions and demonstrate the difference in the main program.**

#include <iostream>

#include <string>

using namespace std;

// Employee class definition

class Employee {

private:

string name;

double salary;

public:

// Constructor

Employee(string n, double s) : name(n), salary(s) {}

// Function to get name

string getName() const {

return name;

}

// Function to get salary

double getSalary() const {

return salary;

}

// Function to set salary

void setSalary(double s) {

salary = s;

}

};

// Function to modify salary of Employee object passed by value

void modifyByValue(Employee emp) {

double newSalary = emp.getSalary() \* 1.1; // Increase salary by 10%

emp.setSalary(newSalary); // Changes are made to the copy, not the original

}

// Function to modify salary of Employee object passed by reference

void modifyByReference(Employee &emp) {

double newSalary = emp.getSalary() \* 1.1; // Increase salary by 10%

emp.setSalary(newSalary); // Changes are made directly to the original object

}

int main() {

// Create an Employee object

Employee emp("John Doe", 50000.0);

// Display initial details

cout << "Before modifications:" << endl;

cout << "Name: " << emp.getName() << ", Salary: $" << emp.getSalary() << endl;

// Demonstrate modifyByValue function

modifyByValue(emp);

cout << "After modifyByValue:" << endl;

cout << "Name: " << emp.getName() << ", Salary: $" << emp.getSalary() << endl;

// Demonstrate modifyByReference function

modifyByReference(emp);

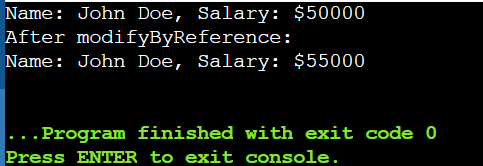
cout << "After modifyByReference:" << endl;

cout << "Name: " << emp.getName() << ", Salary: $" << emp.getSalary() << endl;

return 0;

}

**Output:**

****