**Day-1[24.06.24]**

**Task -1:-** **Write a C program to reverse a given string without using any additional library functions.**

#include <stdio.h>

void revStr(char\* str)

{

int length = 0;

int i;

char temp;

for (length = 0; str[length] != '\0'; length++);

for (i = 0; i < length / 2; i++)

{

temp = str[i];

str[i] = str[length - i - 1];

str[length - i - 1] = temp;

}

}

int main()

{

char str[100];

printf("Enter a string: ");

scanf("%s", str);

revStr(str);

printf("Reversed string: %s\n", str);

return 0;

       }

**Task-3**:- **Explain the concept of structures in C and write a program to store student information (name, roll number, marks) using a structure.**

In C, a structure is a user-defined data type that allows the combination of data items of different kinds. Structures are used to group together different types of variables under a single name. Each variable within a structure is called a member.

The syntax to define a structure is:

struct structure\_name {

data\_type member1;

data\_type member2;

...

};

#include <stdio.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

int main() {

struct Student student;

printf("Enter student name: ");

scanf("%s", student.name);

printf("Enter roll number: ");

scanf("%d", &student.rollNumber);

printf("Enter marks: ");

scanf("%f", &student.marks);

printf("\nStudent Information:\n");

printf("Name: %s\n", student.name);

printf("Roll Number: %d\n", student.rollNumber);

printf("Marks: %.2f\n", student.marks);

    return 0;

}**Task-4:- Differentiate between single-linked lists and doubly-linked lists in C. Write code snippets to create a node and perform a basic insertion operation in a singly-linked list.**

Differences Between Singly-Linked Lists and Doubly-Linked Lists in C

|  |  |
| --- | --- |
| **Singly-linked Lists** | **Doubly-linked Lists** |
| Each node contains data and a pointer to the next node. | Each node contains data, a pointer to the next node, and a pointer to the previous node. |
| Navigation is only possible in one direction (from the head to the tail). | Navigation is possible in both directions (forward and backward). |
| Less memory usage compared to doubly-linked lists because each node has only one pointer. | More memory usage because each node has two pointers. |
| Insertion and deletion operations are simpler and slightly faster since only one pointer needs to be updated. | Insertion and deletion operations are more complex as both next and previous pointers need to be updated. |

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

**Task-5**:- **Explain the concept of pointers in C and write a program to swap the values of two variables using pointers.**

A pointer is a variable that stores the memory address of another variable. Pointers provide a powerful and flexible way to manipulate data and memory, allowing for dynamic memory allocation, efficient array handling, and the creation of complex data structures like linked lists and trees.

**Key Concepts:**

1. **Declaration**: A pointer is declared using the asterisk (\*) symbol. For example, int \*ptr; declares a pointer to an integer.
2. **Address-of Operator (&)**: The address-of operator is used to get the memory address of a variable. For example, ptr = &var; assigns the address of var to the pointer ptr.
3. **Dereference Operator (\*)**: The dereference operator is used to access the value stored at the memory address pointed to by the pointer. For example, \*ptr gives the value of the variable that ptr points to.
4. **Pointer Arithmetic**: Pointers can be incremented or decremented to point to the next or previous memory location.

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x, y;

printf("Enter value for x: ");

scanf("%d", &x);

printf("Enter value for y: ");

scanf("%d", &y);

printf("Before swapping: x = %d, y = %d\n", x, y);

swap(&x, &y);

printf("After swapping: x = %d, y = %d\n", x, y);return 0;

}

**Pointer Arithmetic:**

1. **Write a C program to create an integer array of size 5, initialize it with values from 1 to 5, and then use pointer arithmetic to print each element of the array.**

#include <stdio.h>

int main() {

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

int \*ptr = arr;

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

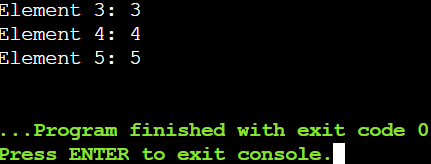
printf("Element %d: %d\n", i+1, \*(ptr + i));

}

return 0;

}

Output



**Pointer to Pointer:**

1. **Write a C program to create a pointer to a pointer for an integer variable. Initialize the integer variable with a value, and then print its value using both the single pointer and the pointer to pointer.**

#include <stdio.h>

int main() {

int num = 10;

int \*ptr = &num;

int \*\*ptr\_to\_ptr = &ptr;

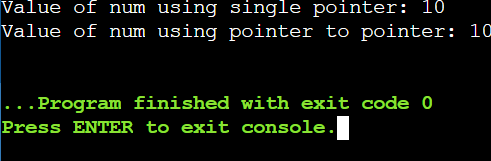
printf("Value of num using single pointer: %d\n", \*ptr);

printf("Value of num using pointer to pointer: %d\n", \*\*ptr\_to\_ptr);

return 0;

}

Output



**Pointer Function Parameters:**

1. **Write a C function void swap(int \*a, int \*b) that swaps the values of two integers. Then, write a main function to test this swap function using pointer arguments.**

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 5, y = 10;

printf("Before swap:\n");

printf("x = %d\n", x);

printf("y = %d\n", y);

swap(&x, &y);

printf("\nAfter swap:\n");

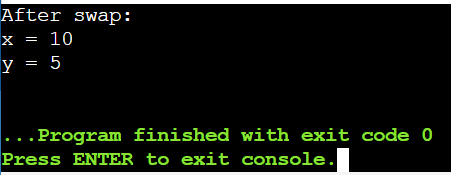
printf("x = %d\n", x);

printf("y = %d\n", y);

return 0;

}

Ouput



**Dynamic Memory Allocation:**

1. **Write a C program to dynamically allocate memory for an array of integers of size 10. Initialize the array with values from 1 to 10, then print the values and free the allocated memory.**

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr;

int size = 10;

int i;

arr = (int \*)malloc(size \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed. Exiting...\n");

return 1;

}

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

arr[i] = i + 1;

}

printf("Array elements:\n");

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

printf("%d ", arr[i]);

}

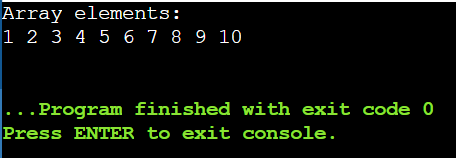
printf("\n");

free(arr);

return 0;

}

Output



**Pointer to Function:**

1. **Write a C program to create a function pointer that points to a function int add(int, int). Use the function pointer to call the add function and print the result.**

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int main() {

int (\*ptr\_add)(int, int);

ptr\_add = add;

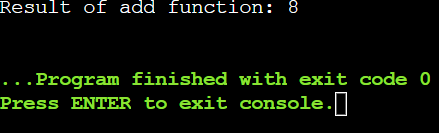
int result = ptr\_add(5, 3);

printf("Result of add function: %d\n", result);

return 0;

}

Ouput



**Functions**

**Recursive Function:**

1. **Write a C function int factorial(int n) that calculates the factorial of a given number using recursion. Test this function in the main program by calculating and printing the factorial of 5.**

#include <stdio.h>

int factorial(int n) {

if (n == 0) {

return 1; // Base case: factorial of 0 is 1

} else {

return n \* factorial(n - 1); // Recursive case

}

}

int main() {

int number = 5;

int result;

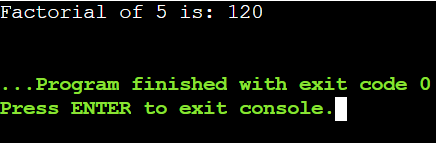
result = factorial(number);

printf("Factorial of %d is: %d\n", number, result);

return 0;

}

Ouput



**Array of Function Pointers:**

1. **Write a C program to create an array of function pointers, where each function takes two integers as arguments and returns an integer. Include functions for addition, subtraction, multiplication, and division. Use the array to perform these operations on two integers and print the results.**

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

int multiply(int a, int b) {

return a \* b;

}

int divide(int a, int b) {

if (b != 0) {

return a / b;

} else {

printf("Error: Division by zero\n");

return 0;

}

}

int main() {

int (\*operations[4])(int, int) = {add, subtract, multiply, divide};

int x = 10, y = 5;

int result;

for (int i = 0; i < 4; i++) {

result = operations[i](x, y);

switch (i) {

case 0:

printf("Addition: %d + %d = %d\n", x, y, result);

break;

case 1:

printf("Subtraction: %d - %d = %d\n", x, y, result);

break;

case 2:

printf("Multiplication: %d \* %d = %d\n", x, y, result);

break;

case 3:

printf("Division: %d / %d = %d\n", x, y, result);

break;

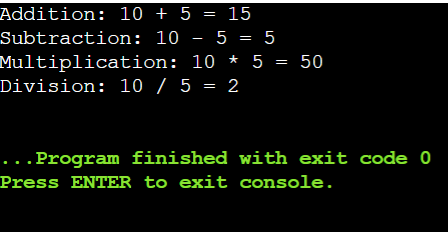
}

}

return 0;

}

Ouput



**Function Returning Pointer:**

1. **Write a C function int\* createArray(int size) that dynamically allocates an array of integers of the given size and returns a pointer to the array. Initialize the array with values from 1 to size and print the array in the main function.**

#include <stdio.h>

#include <stdlib.h>

int\* createArray(int size) {

int\* array = (int\*)malloc(size \* sizeof(int));

if (array == NULL) {

printf("Memory allocation failed\n");

exit(1); // Exit the program if memory allocation failed

}

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

array[i] = i + 1;

}

return array;

}

int main() {

int size;

printf("Enter the size of the array: ");

scanf("%d", &size);

int\* array = createArray(size);

// Print the array

printf("The array is: ");

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

printf("%d ", array[i]);

}

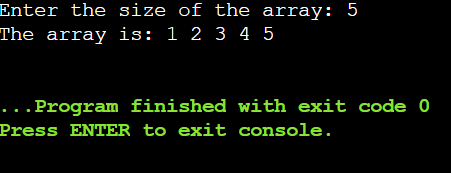
printf("\n");

free(array);

return 0;

}

Output



**Higher-Order Functions:**

1. **Write a C function void applyFunction(int arr[], int size, void (\*func)(int \*)) that takes an array, its size, and a pointer to a function that operates on each element of the array. Write a sample function to double the value of each element and use applyFunction to apply it to an array.**

**Static Variables in Functions:**

1. **Write a C function that uses a static variable to count how many times the function has been called. Test this function in the main program by calling it multiple times and printing the count.**

**Structures**

**Structure Basics:**

1. **Define a structure struct Point with two integer members x and y. Write a C program to create a Point variable, initialize it with values, and print the values.**

#include <stdio.h>

struct Point {

int x;

int y;

};

int main() {

struct Point p;

p.x = 10;

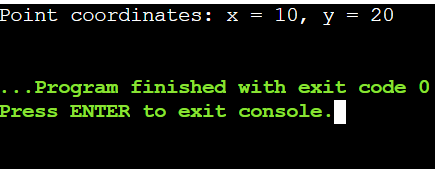
p.y = 20;

printf("Point coordinates: x = %d, y = %d\n", p.x, p.y);

return 0;

}

Ouput



**Array of Structures:**

1. **Write a C program to define a structure struct Student with members name, age, and marks. Create an array of 3 students, initialize them with values, and print the details of each student.**

#include <stdio.h>

struct Student {

char name[50];

int age;

float marks;

};

int main() {

struct Student student[3];

snprintf(student[0].name, sizeof(student[0].name), "Mahesh");

student[0].age = 24;

student[0].marks = 80;

snprintf(student[1].name, sizeof(student[1].name), "Prabhas");

student[1].age = 23;

student[1].marks = 91;

snprintf(student[2].name, sizeof(student[2].name), "Tarak");

student[2].age = 18;

student[2].marks = 100;

for (int i = 0; i < 3; i++) {

printf("Student %d:\n", i + 1);

printf("Name: %s\n", student[i].name);

printf("Age: %d\n", student[i].age);

printf("Marks: %.2f\n", student[i].marks);

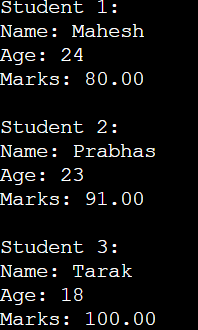
printf("\n");

}

return 0;

}

Ouput



**Structure Pointers:**

1. **Write a C program to define a structure struct Rectangle with members length and width. Create a pointer to a Rectangle variable, dynamically allocate memory for it, initialize the members, and print the values.**

**Nested Structures:**

1. **Write a C program to define a structure struct Date with members day, month, and year, and another structure struct Student with members name and birthdate of type struct Date. Create a Student variable, initialize it with values, and print the student's details including the birthdate.**

#include <stdio.h>

struct Date {

int day;

int month;

int year;

};

struct Student {

char name[50];

struct Date birthdate;

};

int main() {

struct Student student;

snprintf(student.name, sizeof(student.name), "Mahesh");

student.birthdate.day = 16;

student.birthdate.month = 12;

student.birthdate.year = 2000;

printf("Student Details:\n");

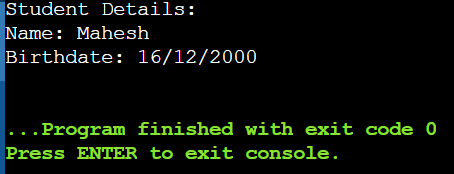
printf("Name: %s\n", student.name);

printf("Birthdate: %02d/%02d/%04d\n", student.birthdate.day, student.birthdate.month, student.birthdate.year);

return 0;

}

Output



**Structure as Function Argument:**

1. **Write a C function void printStudent(struct Student s) that takes a Student structure as an argument and prints the details. Define a Student structure in the main program, initialize it with values, and call printStudent to print the student's details.**

#include <stdio.h>

struct Student {

char name[50];

int age;

float marks;

};

void printStudent(struct Student s) {

printf("Student Details:\n");

printf("Name: %s\n", s.name);

printf("Age: %d\n", s.age);

printf("Marks: %.2f\n", s.marks);

}

int main() {

struct Student student;

snprintf(student.name, sizeof(student.name), "Mahesh");

student.age = 24;

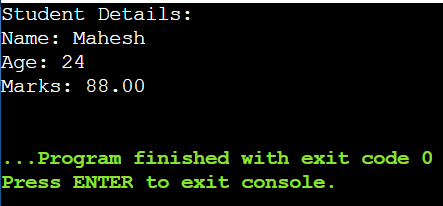
student.marks = 88;

printStudent(student);

return 0;

}

Output



1. #include <iostream>

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);

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

return 0;

}

1. #include <iostream>

using namespace std;

class Point{

public:

double x,y;

};

void offset(Point &p, int x, int y)

{

p.x += x;

p.y += y;

cout << "inside of the object"<<&p<< endl;

}

int main() {

Point p;

p.x = 3.0;

p.y = 4.0;

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

offset(p,1.0,2.0);

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

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

return 0;

}

1. #include <iostream>

using namespace std;

class Point{

public:

double x;

double y;

/\* Point(){

x =0.0;

y =0.0;

cout<< "constructor called"<<endl;

}\*/

Point(double p, double q){ //parameterized constructor

x = p;

y = q;

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

}

};

void offset(Point &p, int x, int y)

{

p.x += x;

p.y += y;

cout << "inside address of the 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 of the object"<<&p<< endl;

offset(p,1.0,2.0);

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

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

return 0;

}

1. #include <iostream>

using namespace std;

class Rectangle {

private:

double length;

double width;

public:

// setter methods

void setLength(double l) {

length = l;

}

void setWidth(double w) {

width = w;

}

// Getter methods

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;

}

1. #include <iostream>

using namespace std;

class Animal {

public:

void eat() {

cout << "This animal is eating." << endl;

}

};

class Dog : public Animal {

public:

void bark () {

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

}

};

int main() {

Dog myDog;

myDog.eat(); //Inherited method

myDog.bark(); // Child class method

return 0;

}

**Day-2[25.06.24]**

**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>

#include <string>

using namespace std;

class Student {

private:

string name;

int age;

public:

// Default constructor

Student() {

name = "Unknown";

age = 0;

}

// Method to display student's details

void display() const {

cout << "Name: " << name << "\n";

cout << "Age: " << age << "\n";

}

};

int main() {

// Create a Student object using the default constructor

Student student;

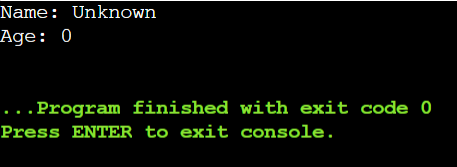
// Display the student's details

student.display();

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

class Rectangle {

private:

double length;

double width;

public:

// Parameterized constructor

Rectangle(double l, double w) {

length = l;

width = w;

}

// Method to calculate area

double area() {

return length \* width;

}

};

int main() {

// Create a Rectangle object with length 5.0 and width 3.0

Rectangle rect(5.0, 3.0);

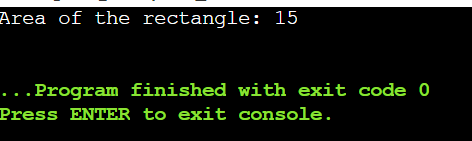
// Calculate and print the area of the rectangle

cout << "Area of the rectangle: " << rect.area() << 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

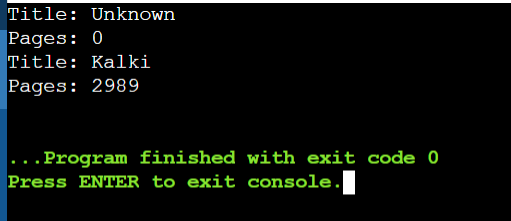
Book existingBook("Kalki", 2989);

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>

class Complex {

private:

double real;

double imaginary;

public:

// Default constructor

Complex() {

real = 0.0;

imaginary = 0.0;

}

// Parameterized constructor

Complex(double r, double i) {

real = r;

imaginary = i;

}

// Method to display complex number

void display() const {

std::cout << "Complex number: " << real << " + " << imaginary << "i\n";

}

};

int main() {

// Create a Complex object using the default constructor

Complex c1;

c1.display();

// Create a Complex object using the parameterized constructor

Complex Complexes(3.5, 2.5); // Using Complexes as the object name

Complexes.display();

// Create another Complex object using different naming

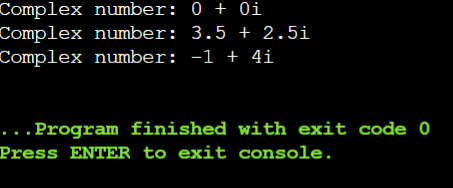
Complex moreComplexes(-1.0, 4.0); // Using moreComplexes as the object name

moreComplexes.display();

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;

// Function to increment the value pointed to by ptr

void increment(int\* 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

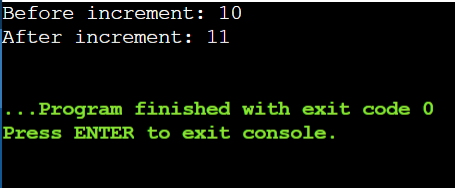
// Call the increment function with the address of num

increment(&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

class Circle {

private:

double radius;

public:

// Constructor

Circle(double r) {

radius = r;

}

// Method to calculate area

double area() {

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

}

};

int main() {

// Create an object of Circle

Circle c(5.0);

// Create a pointer to the Circle object

Circle\* ptrCircle = &c;

// Call the area method using the pointer

double circleArea = ptrCircle->area();

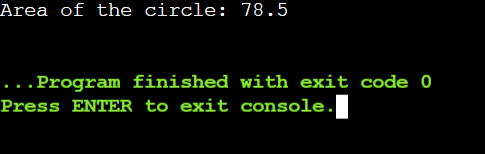
// Print the area of the circle

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() {

// Define an array of integers

int values[] = {10, 20, 30, 40, 50};

// Define an array of pointers to 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] << " ";

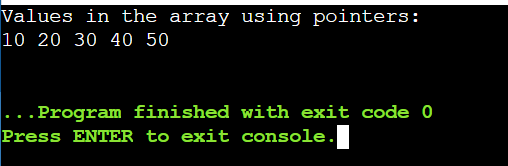
}

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() {

// Define an array of integers

int arr[] = {70, 80, 90, 100, 110};

// Calculate the size of the array

int size = sizeof(arr) / sizeof(arr[0]);

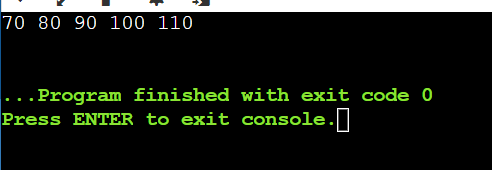
// Call the function to print the array elements

printArray(arr, size);

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() {

// Allocate memory for an integer

int\* ptr = new int;

// Assign a value to the allocated memory

\*ptr = 42;

// Print the value

cout << "Value stored at ptr: " << \*ptr << endl;

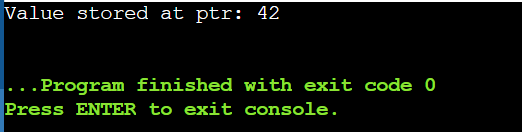
// Free the allocated memory

delete ptr;

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;

// Function to swap two integers using references

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

int temp = a;

a = b;

b = temp;

}

int main() {

// Declare and initialize two integers

int num1 = 5;

int num2 = 10;

// Print initial values

cout << "Before swapping:" << endl;

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

// Call the swap function

swap(num1, num2);

// Print values after swapping

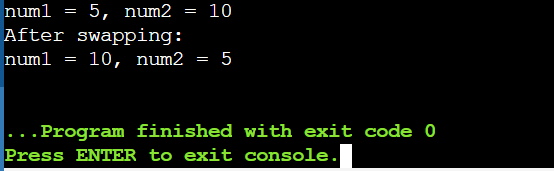
cout << "After swapping:" << endl;

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

class Box {

private:

double length;

double width;

double height;

public:

// Constructor

Box(double l, double w, double h) {

length = l;

width = w;

height = h;

}

// Method to calculate volume

double volume() {

return length \* width \* height;

}

};

int main() {

// Create an object of class Box

Box myBox(3.0, 4.0, 5.0);

// Create a reference to the Box object

Box& boxRef = myBox;

// Call the volume method using the reference

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

return 0;

}

**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 and return reference to the largest element in an array

int& findLargest(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 arr[] = {10, 5, 8, 15, 3};

// Call the function to find the reference to the largest element

int& largest = findLargest(arr, 5);

// Print the largest element using the reference

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

// Modify the largest element through the reference

largest = 20;

// Print the modified array

cout << "Modified array:" << endl;

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

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

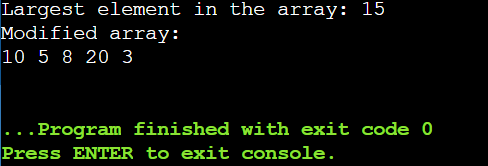
}

cout << endl;

return 0;

}

**Output**

****

**### Pass by Value and Reference**

**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;

}

int main() {

int number = 5;

// Print initial value of number

cout << "Before calling addTen: " << number << endl;

// Call addTen function

addTen(number);

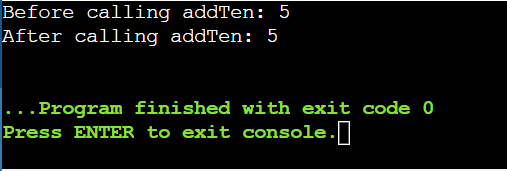
// Print value of number after function call

cout << "After calling addTen: " << 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 reference

void addTenRef(int &num) {

num += 10;

}

int main() {

int number = 5;

// Print initial value of number

cout << "Before calling addTenRef: " << number << endl;

// Call addTenRef function

addTenRef(number);

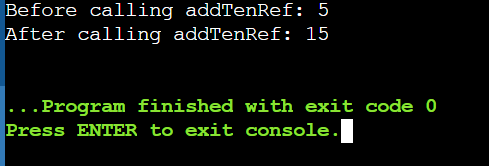
// Print value of number after function call

cout << "After calling addTenRef: " << 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;

// Function to return a reference to a static variable

int& staticVariable() {

static int num = 10; // Static variable initialization

return num; // Return reference to static variable

}

int main() {

// Get reference to the static variable

int &ref = staticVariable();

// Print initial value

cout << "Initial value: " << ref << endl;

// Modify the value through the reference

ref = 20;

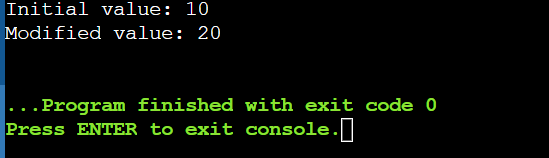
// Print modified value

cout << "Modified value: " << ref << 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("Mahesh", 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);

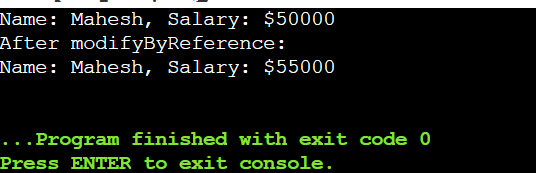
cout << "After modifyByReference:" << endl;

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

return 0;

}

**Output**

****