1. Write a program to print the address of a variable using pointer.

**Input:**

An integer variable num is initialized with a value (e.g., 42).

**Process:**

Store the address of num in a pointer ptr.

Use the pointer to access and display the memory address.

**Output:**

Print the value of num.

Print the address of num using &num.

Print the address of num using the pointer ptr.

#include <stdio.h>

int main() {

int num = 42;

int \*ptr;

ptr = &num;

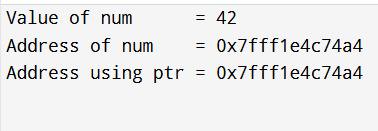
printf("Value of num = %d\n", num);

printf("Address of num = %p\n", &num);

printf("Address using ptr = %p\n", ptr);

return 0;

}



1. Write a program to access array elements using pointers.

**Input:**

* An array of integers: {10, 20, 30, 40, 50}

**🔹 Process:**

* Use a pointer to traverse the array.
* Access each element using pointer arithmetic: \*(ptr + i)

**🔹 Output:**

* Display each element of the array using the pointer.

#include <stdio.h>

int main() {

int arr[5] = {10, 20, 30, 40, 50};

int \*ptr;

ptr = arr;

printf("Accessing array elements using pointer:\n");

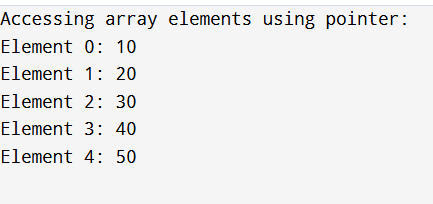
for (int i = 0; i < 5; i++) {

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

}

return 0;

}



1. Write a program to swap two numbers using pointers.

**Input:**

* Two integers: num1 = 5, num2 = 10

**🔹 Process:**

* Pass addresses of num1 and num2 to the swap function.
* Inside the function, use a pointer to swap the values.

**🔹 Output:**

* Display values of num1 and num2 **before** and **after** swapping.

#include <stdio.h>

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

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int num1 = 5, num2 = 10;

printf("Before swapping:\n");

printf("num1 = %d, num2 = %d\n", num1, num2);

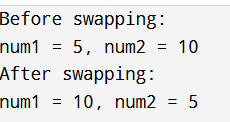
swap(&num1, &num2);

printf("After swapping:\n");

printf("num1 = %d, num2 = %d\n", num1, num2);

return 0;

}



1. Write a program to add two numbers using pointers.

**Input:**

* Two integer values: num1 = 10, num2 = 20

**🔹 Process:**

* Use two pointers ptr1 and ptr2 to store addresses of num1 and num2.
* Add values pointed to by ptr1 and ptr2.

**🔹 Output:**

* Print the sum of the two numbers.

#include <stdio.h>

int main() {

int num1 = 10, num2 = 20;

int \*ptr1, \*ptr2;

int sum;

ptr1 = &num1;

ptr2 = &num2;

sum = \*ptr1 + \*ptr2;

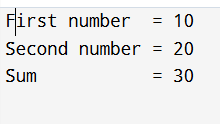
printf("First number = %d\n", \*ptr1);

printf("Second number = %d\n", \*ptr2);

printf("Sum = %d\n", sum);

return 0;

}



1. Write a program to find the length of a string using pointers.

**Input:**

A string: "Hello, world!"

**🔹 Process:**

Use a pointer to traverse the string character by character.

Count characters until the null terminator \0 is reached.

**🔹 Output:**

Display the length of the string (number of characters before \0).

#include <stdio.h>

int main() {

char str[] = "Hello, world!";

char \*ptr = str;

int length = 0;

while (\*ptr != '\0') {

length++;

ptr++;

}

printf("The length of the string is: %d\n", length);

return 0;

}



1. Write a program to reverse a string using pointers.

**Input:**

* A string: "Hello, world!"

**🔹 Process:**

* Use two pointers (start and end) to swap characters from both ends.
* Move pointers towards the center until they meet.

**🔹 Output:**

* Print the reversed string.

#include <stdio.h>

void reverseString(char \*str) {

char \*start = str;

char \*end = str;

while (\*end != '\0') {

end++;

}

end--;

while (start < end) {

char temp = \*start;

\*start = \*end;

\*end = temp;

start++;

end--;

}

}

int main() {

char str[] = "Hello, world!";

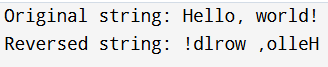
printf("Original string: %s\n", str);

reverseString(str);

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

return 0;

}



1. Write a program to count vowels using pointer.

**Input:**

* A string: "Hello, World!"

**🔹 Process:**

* Use a pointer to traverse the string.
* Check each character if it's a vowel (a, e, i, o, u), case-insensitive.
* Count the vowels.

**🔹 Output:**

* Print the total number of vowels found.

#include <stdio.h>

int countVowels(char \*str) {

int count = 0;

char \*ptr = str;

while (\*ptr != '\0') {

char ch = \*ptr;

if (ch >= 'A' && ch <= 'Z') {

ch = ch + 32; // 'A'->'a', 'B'->'b', etc.

}

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

count++;

}

ptr++;

}

return count;

}

int main() {

char str[] = "Hello, World!";

int vowels = countVowels(str);

printf("Number of vowels in \"%s\" = %d\n", str, vowels);

return 0;

}



1. Write a program to demonstrate pointer to pointer.

**Input:**

* An integer variable var = 100

**🔹 Process:**

* Declare a pointer ptr to hold the address of var.
* Declare a pointer to pointer pptr to hold the address of ptr.
* Access the value of var via ptr and via pptr.

**🔹 Output:**

* Print the value of var directly, via ptr, and via pptr.
* Print the addresses stored and addresses of the pointers.

#include <stdio.h>

int main() {

int var = 100;

int \*ptr;

int \*\*pptr;

ptr = &var;

pptr = &ptr;

printf("Value of var: %d\n", var);

printf("Value via ptr: %d\n", \*ptr);

printf("Value via pptr: %d\n", \*\*pptr);

printf("Address of var: %p\n", &var);

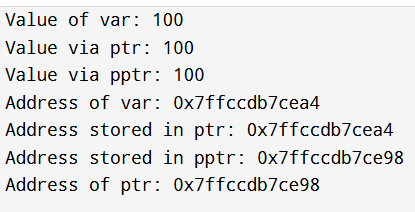
printf("Address stored in ptr: %p\n", ptr);

printf("Address stored in pptr: %p\n", pptr);

printf("Address of ptr: %p\n", &ptr);

return 0;

}



1. Write a program to allocate memory using malloc() and free it.

**Input:**

* Number of integers to allocate memory for: n = 5

**🔹 Process:**

* Use malloc() to allocate memory dynamically.
* Assign values to the allocated memory.
* Print those values.
* Use free() to release the allocated memory.

**🔹 Output:**

* Display the values stored in the dynamically allocated memory.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr;

int n = 5;

ptr = (int \*)malloc(n \* sizeof(int));

if (ptr == NULL) {

printf("Memory allocation failed!\n");

return 1; // Exit if malloc fails

}

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

ptr[i] = (i + 1) \* 10;

}

printf("Values in allocated memory:\n");

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

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

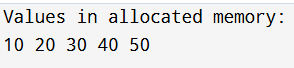
}

printf("\n");

free(ptr);

return 0;

}



1. Write a program to sort an array using pointer notation.

**Input:**

* An array of integers: {50, 20, 40, 10, 30}

**🔹 Process:**

* Use nested loops to compare elements.
* Use pointer arithmetic (\*(arr + i)) to access and swap elements to sort in ascending order.

**🔹 Output:**

* Print the original array.
* Print the sorted array.

#include <stdio.h>

void sortArray(int \*arr, int n)

{

int temp;

for (int i = 0; i < n - 1; i++)

{

for (int j = i + 1; j < n; j++)

{

if (\*(arr + i) > \*(arr + j))

{

temp = \*(arr + i);

\*(arr + i) = \*(arr + j);

\*(arr + j) = temp;

}

}

}

}

int main() {

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

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

printf("Original array:\n");

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

printf("%d ", \*(arr + i));

}

printf("\n");

sortArray(arr, n);

printf("Sorted array:\n");

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

printf("%d ", \*(arr + i));

}

printf("\n");

return 0;

}

