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

IPO

* **Input:** A variable with a value (entered by the user or predefined).
* **Process:** Store the address of the variable in a pointer and print it.
* **Output:** Address of the variable.

CODE

#include <stdio.h>

int main() {

int a = 10;

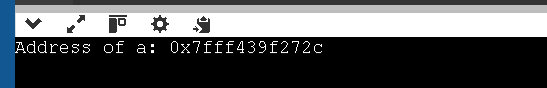
int \*ptr = &a;

printf("Address of a: %p\n", (void\*)ptr);

return 0;

}

OUTPUT



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

IPO

**Input:** Array elements from the user.

**Process:** Use a pointer to iterate through the array and access elements.

**Output:** Display the array elements using pointers.

CODE

#include <stdio.h>

int main() {

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

int \*ptr = arr;

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

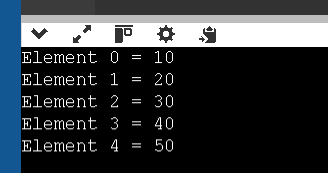
printf("Element %d = %d\n", i, \*(ptr + i));

}

return 0;

}

OUTPUT



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

IPO

**Input:** Two numbers from the user.

**Process:** Use pointers to swap the values without using a third variable.

**Output:** Numbers after swapping.

CODE

#include <stdio.h>

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

int main() {

int a = 5, b = 10;

printf("Before Swap: a=%d, b=%d\n", a, b);

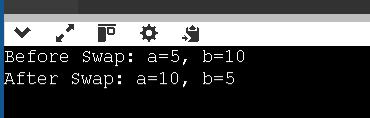
swap(&a, &b);

printf("After Swap: a=%d, b=%d\n", a, b);

return 0;

}

OUTPUT



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

IPO

**Input:** Two numbers from the user.

**Process:** Store the numbers in variables, use pointers to access them, and add the values.

**Output:** Sum of the two numbers.

CODE

#include <stdio.h>

int main() {

int a = 7, b = 8, sum;

int \*p1 = &a, \*p2 = &b;

sum = \*p1 + \*p2;

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

return 0;

}

OUTPUT



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

IPO

**Input:** A string from the user.

**Process:** Use a pointer to traverse the string until the null character is reached, counting characters.

**Output:** Length of the string.

CODE

#include <stdio.h>

int main() {

char str[] = "Hello";

char \*ptr = str;

int length = 0;

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

length++;

ptr++;

}

printf("Length of string = %d\n", length);

return 0;

}

OUPUT



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

IPO

**Input:** A string from the user.

**Process:** Use two pointers (start and end) to swap characters until the middle is reached.

**Output:** Reversed string.

CODE

#include <stdio.h>

#include <string.h>

int main() {

char str[] = "Pointer";

char \*start = str;

char \*end = str + strlen(str) - 1;

char temp;

while (start < end) {

temp = \*start;

\*start = \*end;

\*end = temp;

start++;

end--;

}

printf("Reversed String = %s\n", str);

return 0;

}

OUTPUT



1. Write a program to count vowels using pointer.

IPO

* **Input:** A string from the user.
* **Process:** Use a pointer to traverse the string, checking each character if it’s a vowel, and count them.
* **Output:** Number of vowels in the string.

CODE

#include <stdio.h>

#include <ctype.h>

int main() {

char str[] = "Programming in C";

char \*ptr = str;

int count = 0;

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

char ch = tolower(\*ptr);

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

count++;

ptr++;

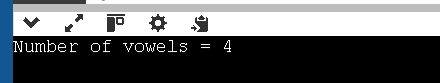
}

printf("Number of vowels = %d\n", count);

return 0;

}

OUTPUT



1. Write a program to demonstrate pointer to pointer.

IPO

**Input:** A variable with a value.

**Process:** Store the variable’s address in a pointer, then store that pointer’s address in another pointer (pointer to pointer), and display values at different levels of dereferencing.

**Output:** Original value, address stored in pointer, and address of pointer.

CODE

#include <stdio.h>

int main() {

int a = 42;

int \*p = &a;

int \*\*pp = &p;

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

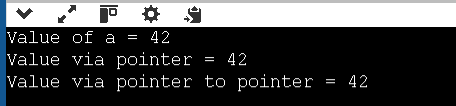
printf("Value via pointer = %d\n", \*p);

printf("Value via pointer to pointer = %d\n", \*\*pp);

return 0;

}

OUTPUT



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

IPO

**Input:** Number of elements to store.

**Process:** Use malloc() to allocate memory dynamically, accept values from the user, display them, then release the memory using free().

**Output:** Entered values and confirmation that memory is freed.

CODE

#include <stdio.h>

#include <stdlib.h>

int main() {

int n;

printf("Enter number of elements: ");

scanf("%d", &n);

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

if (arr == NULL) {

printf("Memory allocation failed\n");

return 1;

}

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

arr[i] = i + 1;

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

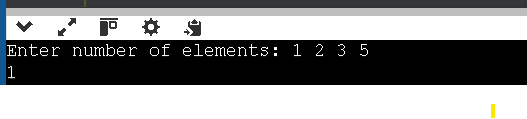
}

free(arr); // Free allocated memory

return 0;

}

OUTPUT



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

IPO

**Input:** Array elements from the user.

**Process:** Use pointer arithmetic to compare and swap elements to sort the array in ascending/descending order.

**Output:** Sorted array

CODE

#include <stdio.h>

void sort(int \*arr, int n) {

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

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

if (\*(arr + j) < \*(arr + i)) {

int temp = \*(arr + i);

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

\*(arr + j) = temp;

}

}

}

}

int main() {

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

int n = 5;

sort(arr, n);

printf("Sorted array: ");

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

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

}

return 0;

}

OUTPUT

