Exercise

Q1. Develop a C program to work with a string using a double pointer. Begin by declaring an array of characters (a string) with a size of at least 6. Declare a double pointer and make it point to the first character of the array. Utilize the double pointer and pointer arithmetic to change the third character of the array to 'X'. Print the modified string.

Example:

Original String: abcdef

Modified String: abXdef

Q2. Develop a C program to manage a list of student roll numbers using dynamic memory allocation. Initially, allocate memory for the array to store roll numbers of 3 students. Prompt the user to enter the roll numbers of the first three students. Print the list of student roll numbers. Use realloc to increase the size of the array to accommodate 5 students. Add roll numbers for two more students to the list. Print the updated list of student roll numbers. Finally, free the memory allocated for the array.

Example:

Enter the initial size of the array: 3

Enter the roll numbers of the first 3 students:

101 102 103

List of student roll numbers:

101 102 103

Enter the number of additional students: 2

Enter the roll numbers for the additional students:

104 105

Updated list of student roll numbers:

101 102 103 104 105

Q3. Develop a C program that calculates the sum of all the elements in an array using pointers. Begin by prompting the user to enter the size of the array. Allocate memory dynamically based on the user's input size. Prompt the user to enter the elements of the array. Implement a function, CalculateSum, that takes a pointer to the array and its size as arguments and calculates the sum of all elements using pointer arithmetic.

int CalculateSum(int *arr, int size);

In the main() function, call the CalculateSum function and print the result.

Example:

Copy code

Enter the size of the array: 5

Enter 5 integers for the array:

2 5 8 1 4

Sum of array elements: 20

Q4. Create a C program that dynamically allocates memory for a user-defined size 2D array using calloc. Prompt the user to enter the elements of the array. Implement a function, PrintTranspose, that takes the 2D array and its dimensions and prints the transpose of the array. Finally, free the memory allocated for the 2D array.

void PrintTranspose(int **matrix, int rows, int cols);

The PrintTranspose function should print the transpose of the given 2D array. In the main() function, prompt the user to enter the number of rows and columns for the 2D array. Allow the user to input the elements of the array. Then, invoke the PrintTranspose function to display the transpose of the entered array. Finally, free the dynamically allocated memory.

Example:

Enter the number of rows: 3

Enter the number of columns: 4

Enter the elements of the 2D array:

1 2 3 4

5 6 7 8

9 10 11 12

Transpose of the 2D array:

1 5 9

2 6 10

3 7 11

4 8 12

Q5. Develop a C program that incorporates a sorting function, SortFunction, to efficiently sort an array of integers based on the user's specified order. The SortFunction should take three parameters: a pointer to the array, the size of the array, and an integer representing the order of sorting. The order parameter should be 1 for ascending order and 2 for descending order.

void SortFunction(int *arr, int *size, int order);

Within this function, implement the sorting logic to arrange the elements of the array either in ascending or descending order based on the value of the order parameter. In the main() function, prompt the user to enter the size of the array and input the array elements. After that, call the SortFunction to sort the array based on the user's specified order. Finally, print the sorted array to validate the correctness of the sorting algorithm. Ensure dynamic memory allocation for the array based on the user's input size.

Example:

Enter the size of the array: 5

Enter 5 integers: 7 2 5 1 9

Enter the order (1 for ascending, 2 for descending): 1

Sorted Array in Ascending Order: 1 2 5 7 9