COLLEGE KI FILE

## 1. Write a program to (a.) create an array of integers and initialize it at compile-time (b.) create another array of floating values and initialize it at run-time (c.) display the elements of both the arrays with proper headings

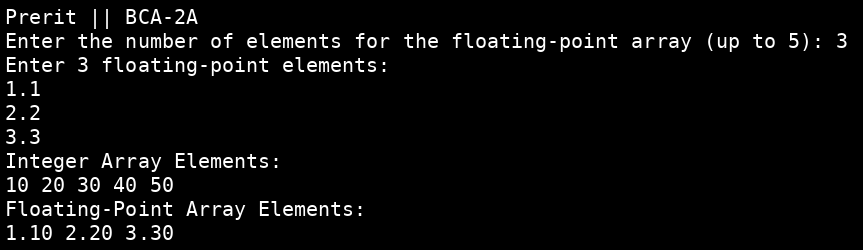
### Algorithm

1) Declare an integer array (e.g., `int intArray[5]`) and initialize it at compile-time with some integer values (e.g., `int intArray[5] = {10, 20, 30, 40, 50};`).  
2) Declare a floating-point array (e.g., `float floatArray[5]`).  
3) Prompt the user to enter the number of elements for the floating-point array (e.g., "Enter the number of elements for the floating-point array:"). Store the input in an integer variable (e.g., `int n`). Ensure `n` is not greater than the declared size of `floatArray`.  
4) Using a `for` loop, prompt the user to enter the elements for the floating-point array. Read each floating-point number using `scanf()` and store it in the respective index of the `floatArray`. The loop should iterate from 0 to `n-1`.  
5) Print a heading for the integer array (e.g., "Integer Array Elements:").  
6) Using a `for` loop, iterate through the integer array and print each element followed by a space.  
7) Print a newline character to separate the outputs.  
8) Print a heading for the floating-point array (e.g., "Floating-Point Array Elements:").  
9) Using a `for` loop, iterate through the floating-point array (from 0 to `n-1`) and print each element, formatted to two decimal places (e.g., using `%.2f`), followed by a space.  
10) Print a newline character.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int intArray[5] = {10, 20, 30, 40, 50};  
 float floatArray[5];  
 int n, i;  
  
 printf("Enter the number of elements for the floating-point array (up to 5): ");  
 scanf("%d", &n);  
  
 printf("Enter %d floating-point elements:\n", n);  
 for (i = 0; i < n; i++) {  
 scanf("%f", &floatArray[i]);  
 }  
  
 printf("Integer Array Elements:\n");  
 for (i = 0; i < 5; i++) {  
 printf("%d ", intArray[i]);  
 }  
 printf("\n");  
  
 printf("Floating-Point Array Elements:\n");  
 for (i = 0; i < n; i++) {  
 printf("%.2f ", floatArray[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



## 2. Write a Program to implement Linear Search for (a.) First occurrence of search item (b.) All occurrences of search item

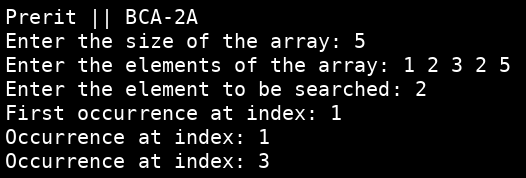
### Algorithm

1) Start  
2) Input the size of the array (n).  
3) Input the elements of the array arr[0] to arr[n-1].  
4) Input the item to be searched (key).  
5) For finding the first occurrence:  
 a) Initialize a variable first\_occurrence = -1.  
 b) Iterate through the array from i = 0 to n-1.  
 c) If arr[i] is equal to key, set first\_occurrence = i and break the loop.  
 d) If first\_occurrence is -1, print "Element not found". Otherwise, print "First occurrence at index: " followed by the value of first\_occurrence.  
6) For finding all occurrences:  
 a) Initialize a variable count = 0.  
 b) Iterate through the array from i = 0 to n-1.  
 c) If arr[i] is equal to key, print "Occurrence at index: " followed by the value of i, and increment count.  
 d) If count is 0, and first\_occurrence was -1, the message "Element not found" would have already been printed. If count is 0 but first\_occurrence was not -1, then it was already found and displayed  
7) Stop

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, key, i, first\_occurrence = -1, count = 0;  
  
 printf("\nEnter the size of the array: ");  
 scanf("%d", &n);  
  
 int arr[n];  
  
 printf("Enter the elements of the array: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 printf("Enter the element to be searched: ");  
 scanf("%d", &key);  
  
 for (i = 0; i < n; i++) {  
 if (arr[i] == key) {  
 first\_occurrence = i;  
 break;  
 }  
 }  
  
 if (first\_occurrence == -1) {  
 printf("Element not found\n");  
 } else {  
 printf("First occurrence at index: %d\n", first\_occurrence);  
 }  
  
 for (i = 0; i < n; i++) {  
 if (arr[i] == key) {  
 printf("Occurrence at index: %d\n", i);  
 count++;  
 }  
 }  
if(first\_occurrence != -1 && count == 0)  
{  
}  
}

### Output:



## 3. Write a program to (a.) create an array of integers and initialize it (b.) Find minimum and maximum elements in the array (c.) Find sum and average of array elements

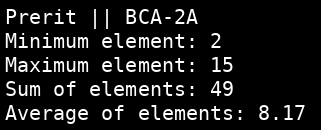
### Algorithm

1) Initialize an integer array `arr` with a predefined set of integer values or by taking input from the user.  
2) Initialize `min` and `max` to the first element of the array, i.e., `min = arr[0]` and `max = arr[0]`.  
3) Initialize `sum` to 0.  
4) Iterate through the array from the second element (index 1) to the last element.  
5) In each iteration:  
 a) Compare the current element `arr[i]` with `min`. If `arr[i]` is less than `min`, update `min` to `arr[i]`.  
 b) Compare the current element `arr[i]` with `max`. If `arr[i]` is greater than `max`, update `max` to `arr[i]`.  
 c) Add the current element `arr[i]` to `sum`.  
6) After the loop finishes, calculate the average by dividing `sum` by the number of elements in the array. Store the result in a float variable `average`.  
7) Print the values of `min`, `max`, `sum`, and `average`.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[] = {12, 5, 8, 2, 15, 7};  
 int size = sizeof(arr) / sizeof(arr[0]);  
 int min = arr[0];  
 int max = arr[0];  
 int sum = 0;  
 float average;  
 for (int i = 0; i < size; i++) {  
 if (arr[i] < min) {  
 min = arr[i];  
 }  
 if (arr[i] > max) {  
 max = arr[i];  
 }  
 sum += arr[i];  
 }  
 average = (float)sum / size;  
 printf("\nMinimum element: %d", min);  
 printf("\nMaximum element: %d", max);  
 printf("\nSum of elements: %d", sum);  
 printf("\nAverage of elements: %.2f", average);  
 return 0;  
}

### Output:



## 4. Write a program to Merge unsorted arrays

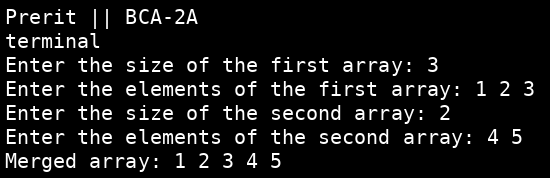
### Algorithm

1) Step 1: Get the sizes of the two unsorted arrays (array1 and array2) from the user.  
2) Step 2: Allocate memory dynamically for the two arrays (array1 and array2) based on their sizes.  
3) Step 3: Get the elements of array1 from the user.  
4) Step 4: Get the elements of array2 from the user.  
5) Step 5: Calculate the size of the merged array (merged\_array\_size = size1 + size2).  
6) Step 6: Allocate memory dynamically for the merged array (merged\_array) based on its size.  
7) Step 7: Copy all elements from array1 to the beginning of merged\_array.  
8) Step 8: Copy all elements from array2 to the end of merged\_array, after the elements of array1.  
9) Step 9: Print the elements of the merged array.  
10) Step 10: Free the dynamically allocated memory for array1, array2, and merged\_array.

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int size1, size2, merged\_array\_size, i, j;  
 int \*array1, \*array2, \*merged\_array;  
  
 printf("\nEnter the size of the first array: ");  
 scanf("%d", &size1);  
  
 array1 = (int \*)malloc(size1 \* sizeof(int));  
 if (array1 == NULL) {  
 printf("Memory allocation failed for array1.\n");  
 return 1;  
 }  
  
 printf("Enter the elements of the first array: ");  
 for (i = 0; i < size1; i++) {  
 scanf("%d", &array1[i]);  
 }  
  
 printf("Enter the size of the second array: ");  
 scanf("%d", &size2);  
  
 array2 = (int \*)malloc(size2 \* sizeof(int));  
 if (array2 == NULL) {  
 printf("Memory allocation failed for array2.\n");  
 free(array1);  
 return 1;  
 }  
  
 printf("Enter the elements of the second array: ");  
 for (i = 0; i < size2; i++) {  
 scanf("%d", &array2[i]);  
 }  
  
 merged\_array\_size = size1 + size2;  
 merged\_array = (int \*)malloc(merged\_array\_size \* sizeof(int));  
 if (merged\_array == NULL) {  
 printf("Memory allocation failed for merged\_array.\n");  
 free(array1);  
 free(array2);  
 return 1;  
 }  
  
 for (i = 0; i < size1; i++) {  
 merged\_array[i] = array1[i];  
 }  
  
 for (j = 0; j < size2; j++) {  
 merged\_array[size1 + j] = array2[j];  
 }  
  
 printf("Merged array: ");  
 for (i = 0; i < merged\_array\_size; i++) {  
 printf("%d ", merged\_array[i]);  
 }  
 printf("\n");  
  
 free(array1);  
 free(array2);  
 free(merged\_array);  
  
 return 0;  
}

### Output:



## 5. Write a program to Marge sorted arrays

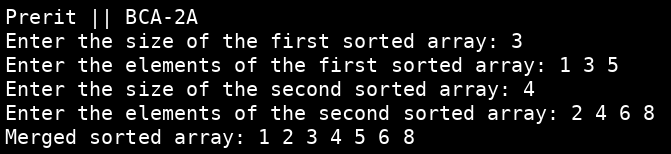
### Algorithm

1) Get the sizes of the two sorted arrays (array1\_size and array2\_size).  
2) Allocate memory for the two sorted arrays (array1 and array2) of their respective sizes.  
3) Allocate memory for the merged array (merged\_array) of size array1\_size + array2\_size.  
4) Get the elements of the first sorted array (array1) from the user.  
5) Get the elements of the second sorted array (array2) from the user.  
6) Initialize three index variables: i = 0 (for array1), j = 0 (for array2), and k = 0 (for merged\_array).  
7) While i < array1\_size and j < array2\_size:  
 a) If array1[i] <= array2[j]:  
 merged\_array[k] = array1[i]  
 i = i + 1  
 b) Else:  
 merged\_array[k] = array2[j]  
 j = j + 1  
 c) k = k + 1  
8) While i < array1\_size:  
 merged\_array[k] = array1[i]  
 i = i + 1  
 k = k + 1  
9) While j < array2\_size:  
 merged\_array[k] = array2[j]  
 j = j + 1  
 k = k + 1  
10) Print the merged array (merged\_array).  
11) Free the allocated memory for array1, array2, and merged\_array.

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int array1\_size, array2\_size, i, j, k;  
 int \*array1, \*array2, \*merged\_array;  
  
 printf("Enter the size of the first sorted array: ");  
 scanf("%d", &array1\_size);  
  
 array1 = (int \*)malloc(array1\_size \* sizeof(int));  
 if (array1 == NULL) {  
 printf("Memory allocation failed.\n");  
 return 1;  
 }  
  
 printf("Enter the elements of the first sorted array: ");  
 for (i = 0; i < array1\_size; i++) {  
 scanf("%d", &array1[i]);  
 }  
  
 printf("Enter the size of the second sorted array: ");  
 scanf("%d", &array2\_size);  
  
 array2 = (int \*)malloc(array2\_size \* sizeof(int));  
 if (array2 == NULL) {  
 printf("Memory allocation failed.\n");  
 free(array1);  
 return 1;  
 }  
  
 printf("Enter the elements of the second sorted array: ");  
 for (i = 0; i < array2\_size; i++) {  
 scanf("%d", &array2[i]);  
 }  
  
 merged\_array = (int \*)malloc((array1\_size + array2\_size) \* sizeof(int));  
 if (merged\_array == NULL) {  
 printf("Memory allocation failed.\n");  
 free(array1);  
 free(array2);  
 return 1;  
 }  
  
 i = 0;  
 j = 0;  
 k = 0;  
  
 while (i < array1\_size && j < array2\_size) {  
 if (array1[i] <= array2[j]) {  
 merged\_array[k] = array1[i];  
 i++;  
 } else {  
 merged\_array[k] = array2[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < array1\_size) {  
 merged\_array[k] = array1[i];  
 i++;  
 k++;  
 }  
  
 while (j < array2\_size) {  
 merged\_array[k] = array2[j];  
 j++;  
 k++;  
 }  
  
 printf("Merged sorted array: ");  
 for (i = 0; i < array1\_size + array2\_size; i++) {  
 printf("%d ", merged\_array[i]);  
 }  
 printf("\n");  
  
 free(array1);  
 free(array2);  
 free(merged\_array);  
  
 return 0;  
}

### Output:



## 6. Write a program to insert a number at a given location in an array.

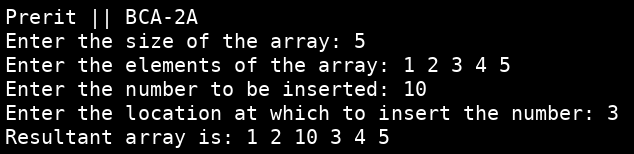
### Algorithm

1) Get the size of the array from the user.  
2) Get the elements of the array from the user.  
3) Get the number to be inserted from the user.  
4) Get the location at which the number is to be inserted from the user.  
5) Shift all the elements from the given location to the end of the array to the right by one position.  
6) Insert the given number at the specified location.  
7) Print the modified array.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[100], size, num, location, i;  
  
 printf("Enter the size of the array: ");  
 scanf("%d", &size);  
  
 printf("Enter the elements of the array: ");  
 for (i = 0; i < size; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 printf("Enter the number to be inserted: ");  
 scanf("%d", &num);  
  
 printf("Enter the location at which to insert the number: ");  
 scanf("%d", &location);  
  
 for (i = size - 1; i >= location - 1; i--) {  
 arr[i + 1] = arr[i];  
 }  
  
 arr[location - 1] = num;  
 size++;  
  
 printf("Resultant array is: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



## 7. Write a program to delete a number from a given location in an array.

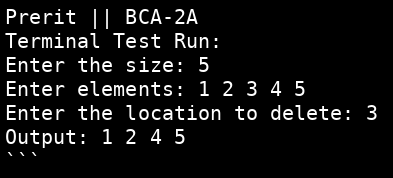
### Algorithm

1) Get the size of the array from the user.  
2) Get the elements of the array from the user.  
3) Get the location from the user where the element needs to be deleted.  
4) Check if the location is valid (i.e., within the bounds of the array). If the location is invalid, print an error message and exit.  
5) Shift all elements from the location to the end of the array one position to the left, effectively overwriting the element at the given location.  
6) Decrement the size of the array by one.  
7) Print the modified array.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int size, location, i;  
  
 scanf("%d", &size);  
  
 int array[size];  
  
 for (i = 0; i < size; i++) {  
 scanf("%d", &array[i]);  
 }  
  
 scanf("%d", &location);  
  
 if (location >= size+1) {  
 printf("Invalid location.\n");  
 } else {  
 for (i = location - 1; i < size - 1; i++) {  
 array[i] = array[i + 1];  
 }  
  
 size--;  
  
 for (i = 0; i < size; i++) {  
 printf("%d ", array[i]);  
 }  
 printf("\n");  
 }  
  
 return 0;  
}

### Output:



## 8. Write a program to search a number in an array and delete it, if found.

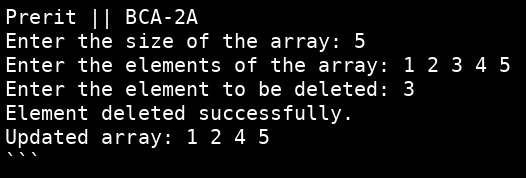
### Algorithm

1) Input the size of the array, `n`.  
2) Input `n` integer elements into the array `arr`.  
3) Input the number to be searched and deleted, `key`.  
4) Initialize `found` to 0 (false).  
5) Iterate through the array `arr` from index `i = 0` to `n-1`.  
6) In each iteration, check if `arr[i]` is equal to `key`.  
7) If `arr[i]` is equal to `key`:  
 a) Set `found` to 1 (true).  
 b) Shift all elements from index `i+1` to `n-1` one position to the left, effectively deleting the element at index `i`. This is done by iterating from `j = i` to `n-2` and setting `arr[j] = arr[j+1]`.  
 c) Decrement the array size `n` by 1.  
 d) Break the loop (since only the first occurence needs to be deleted).  
8) After the loop, check the value of `found`.  
9) If `found` is 1:  
 a) Print "Element deleted successfully."  
 b) Print the updated array elements from index `i = 0` to `n-1`.  
10) If `found` is 0:  
 a) Print "Element not found."

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[100], n, i, j, key, found = 0;  
 printf("Enter the size of the array: ");  
 scanf("%d", &n);  
 printf("Enter the elements of the array: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
 printf("Enter the element to be deleted: ");  
 scanf("%d", &key);  
 for (i = 0; i < n; i++) {  
 if (arr[i] == key) {  
 found = 1;  
 for (j = i; j < n - 1; j++) {  
 arr[j] = arr[j + 1];  
 }  
 n--;  
 break;  
 }  
 }  
 if (found == 1) {  
 printf("Element deleted successfully.\n");  
 printf("Updated array: ");  
 for (i = 0; i < n; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
 } else {  
 printf("Element not found.\n");  
 }  
 return 0;  
}

### Output:



## 9. Write a Program to implement Binary Search

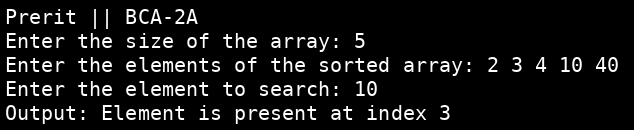
### Algorithm

1) Start.  
2) Get the sorted array (arr) and the target element (x) to search for.  
3) Initialize low = 0 and high = size of the array - 1.  
4) While low is less than or equal to high, repeat steps 5 and 6.  
5) Calculate the middle index: mid = (low + high) / 2.  
6) Compare arr[mid] with x:  
 a) If arr[mid] equals x, return mid (the index of x).  
 b) If arr[mid] is less than x, set low = mid + 1.  
 c) If arr[mid] is greater than x, set high = mid - 1.  
7) If the loop finishes without finding x, return -1 (x is not present in the array).  
8) End.

### Code:

#include <stdio.h>  
  
int binarySearch(int arr[], int low, int high, int x) {  
 while (low <= high) {  
 int mid = low + (high - low) / 2;  
 if (arr[mid] == x)  
 return mid;  
 if (arr[mid] < x)  
 low = mid + 1;  
 else  
 high = mid - 1;  
 }  
 return -1;  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[100], n, x, result;  
 printf("\nEnter the size of the array: ");  
 scanf("%d", &n);  
 printf("Enter the elements of the sorted array: ");  
 for (int i = 0; i < n; i++)  
 scanf("%d", &arr[i]);  
 printf("Enter the element to search: ");  
 scanf("%d", &x);  
 result = binarySearch(arr, 0, n - 1, x);  
 if (result == -1)  
 printf("Element is not present in array");  
 else  
 printf("Element is present at index %d", result);  
 return 0;  
}

### Output:



## 10. Write a Program to perform elimination of duplicate elements from an existing list of elements.

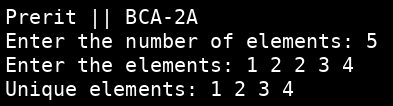
### Algorithm

1) Get the number of elements for the list from the user.  
2) Create an array (list) of the specified size to store the elements.  
3) Prompt the user to enter the elements and store them in the array.  
4) Create a new array (list) to store the unique elements.  
5) Iterate through the original array. For each element:  
 a) Check if the element exists in the unique array.  
 b) If the element does not exist in the unique array, add it to the unique array.  
6) Print the unique elements stored in the unique array.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, i, j, k;  
  
 printf("Enter the number of elements: ");  
 scanf("%d", &n);  
  
 int arr[n];  
 int uniqueArr[n];  
 int uniqueSize = 0;  
  
 printf("Enter the elements: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 for (i = 0; i < n; i++) {  
 int isDuplicate = 0;  
 for (j = 0; j < uniqueSize; j++) {  
 if (arr[i] == uniqueArr[j]) {  
 isDuplicate = 1;  
 break;  
 }  
 }  
 if (!isDuplicate) {  
 uniqueArr[uniqueSize] = arr[i];  
 uniqueSize++;  
 }  
 }  
  
 printf("Unique elements: ");  
 for (i = 0; i < uniqueSize; i++) {  
 printf("%d ", uniqueArr[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



## 11. Create a Matrix. Perform addition, subtraction, Transpose and Multiplication using Switch-Case statement.

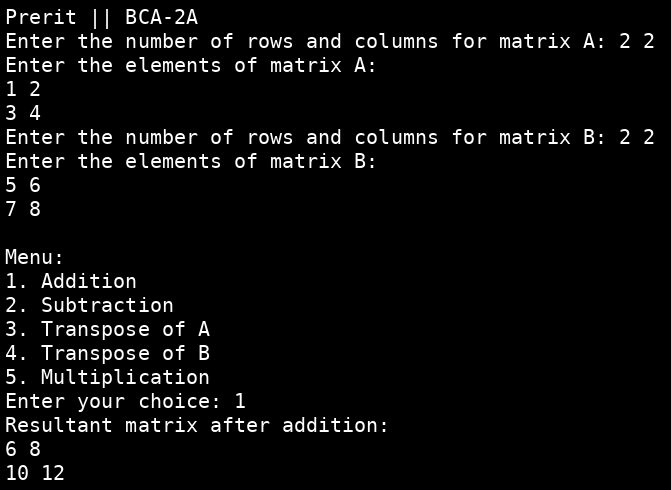
### Algorithm

1) Input the dimensions (rows and columns) of the first matrix (A).  
2) Input the elements of the first matrix (A).  
3) Input the dimensions of the second matrix (B).  
4) Input the elements of the second matrix (B).  
5) Display a menu of operations: Addition, Subtraction, Transpose of A, Transpose of B, Multiplication.  
6) Input the user's choice of operation (1-5).  
7) Use a switch-case statement to perform the selected operation:  
 a) Case 1 (Addition):  
 i) Check if the dimensions of A and B are compatible for addition (rows and columns must be equal). If not, print an error message.  
 ii) If compatible, create a result matrix C with the same dimensions as A and B.  
 iii) Calculate the sum of A and B, storing the result in C (C[i][j] = A[i][j] + B[i][j]).  
 iv) Print the result matrix C.  
 b) Case 2 (Subtraction):  
 i) Check if the dimensions of A and B are compatible for subtraction (rows and columns must be equal). If not, print an error message.  
 ii) If compatible, create a result matrix C with the same dimensions as A and B.  
 iii) Calculate the difference of A and B, storing the result in C (C[i][j] = A[i][j] - B[i][j]).  
 iv) Print the result matrix C.  
 c) Case 3 (Transpose of A):  
 i) Create a result matrix C with dimensions swapped (columns x rows).  
 ii) Calculate the transpose of A, storing the result in C (C[j][i] = A[i][j]).  
 iii) Print the result matrix C.  
 d) Case 4 (Transpose of B):  
 i) Create a result matrix C with dimensions swapped (columns x rows).  
 ii) Calculate the transpose of B, storing the result in C (C[j][i] = B[i][j]).  
 iii) Print the result matrix C.  
 e) Case 5 (Multiplication):  
 i) Check if the number of columns in A is equal to the number of rows in B. If not, print an error message.  
 ii) If compatible, create a result matrix C with dimensions (rows of A x columns of B).  
 iii) Initialize all elements of C to 0.  
 iv) Calculate the product of A and B, storing the result in C (C[i][j] = sum(A[i][k] \* B[k][j])).  
 v) Print the result matrix C.  
 f) Default: Print an error message for invalid choice.  
8) End the program.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int rowsA, colsA, rowsB, colsB, choice, i, j, k;  
 int matrixA[10][10], matrixB[10][10], result[10][10];  
  
 printf("Enter the number of rows and columns for matrix A: ");  
 scanf("%d %d", &rowsA, &colsA);  
  
 printf("Enter the elements of matrix A:\n");  
 for (i = 0; i < rowsA; i++) {  
 for (j = 0; j < colsA; j++) {  
 scanf("%d", &matrixA[i][j]);  
 }  
 }  
  
 printf("Enter the number of rows and columns for matrix B: ");  
 scanf("%d %d", &rowsB, &colsB);  
  
 printf("Enter the elements of matrix B:\n");  
 for (i = 0; i < rowsB; i++) {  
 for (j = 0; j < colsB; j++) {  
 scanf("%d", &matrixB[i][j]);  
 }  
 }  
  
 printf("\nMenu:\n");  
 printf("1. Addition\n");  
 printf("2. Subtraction\n");  
 printf("3. Transpose of A\n");  
 printf("4. Transpose of B\n");  
 printf("5. Multiplication\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 if (rowsA != rowsB || colsA != colsB) {  
 printf("Matrices cannot be added. Dimensions are not compatible.\n");  
 break;  
 }  
 printf("Resultant matrix after addition:\n");  
 for (i = 0; i < rowsA; i++) {  
 for (j = 0; j < colsA; j++) {  
 result[i][j] = matrixA[i][j] + matrixB[i][j];  
 printf("%d ", result[i][j]);  
 }  
 printf("\n");  
 }  
 break;  
  
 case 2:  
 if (rowsA != rowsB || colsA != colsB) {  
 printf("Matrices cannot be subtracted. Dimensions are not compatible.\n");  
 break;  
 }  
 printf("Resultant matrix after subtraction:\n");  
 for (i = 0; i < rowsA; i++) {  
 for (j = 0; j < colsA; j++) {  
 result[i][j] = matrixA[i][j] - matrixB[i][j];  
 printf("%d ", result[i][j]);  
 }  
 printf("\n");  
 }  
 break;  
  
 case 3:  
 printf("Transpose of matrix A:\n");  
 for (i = 0; i < colsA; i++) {  
 for (j = 0; j < rowsA; j++) {  
 printf("%d ", matrixA[j][i]);  
 }  
 printf("\n");  
 }  
 break;  
  
 case 4:  
 printf("Transpose of matrix B:\n");  
 for (i = 0; i < colsB; i++) {  
 for (j = 0; j < rowsB; j++) {  
 printf("%d ", matrixB[j][i]);  
 }  
 printf("\n");  
 }  
 break;  
  
 case 5:  
 if (colsA != rowsB) {  
 printf("Matrices cannot be multiplied. Dimensions are not compatible.\n");  
 break;  
 }  
 printf("Resultant matrix after multiplication:\n");  
 for (i = 0; i < rowsA; i++) {  
 for (j = 0; j < colsB; j++) {  
 result[i][j] = 0;  
 for (k = 0; k < colsA; k++) {  
 result[i][j] += matrixA[i][k] \* matrixB[k][j];  
 }  
 printf("%d ", result[i][j]);  
 }  
 printf("\n");  
 }  
 break;  
  
 default:  
 printf("Invalid choice.\n");  
 }  
  
 return 0;  
}

### Output:



## 12. Read and display Upper, Lower and Tri-diagonal matrices

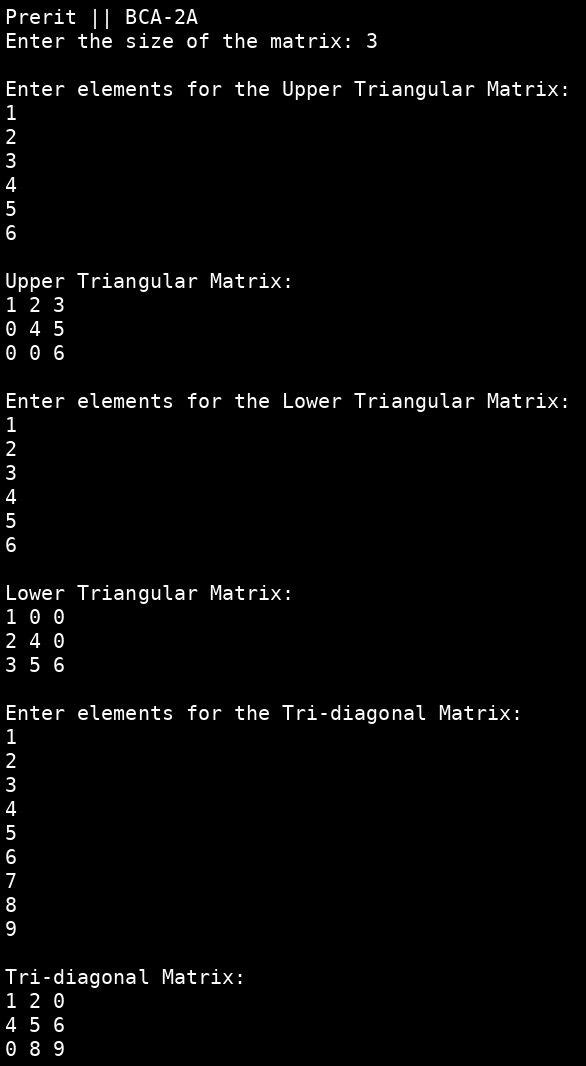
### Algorithm

1) Start.  
2) Input the size (n) of the square matrix.  
3) Declare three 2D arrays: `upper[n][n]`, `lower[n][n]`, and `tridiagonal[n][n]`.  
4) Read the elements of the upper triangular matrix:  
 a) Iterate through rows `i` from 0 to `n-1`.  
 b) Iterate through columns `j` from `i` to `n-1`.  
 c) Input the element `upper[i][j]`.  
 d) For columns `j` from 0 to `i-1`, set `upper[i][j] = 0`.  
5) Display the upper triangular matrix:  
 a) Iterate through rows `i` from 0 to `n-1`.  
 b) Iterate through columns `j` from 0 to `n-1`.  
 c) Print `upper[i][j]` followed by a space.  
 d) Print a newline character after each row.  
6) Read the elements of the lower triangular matrix:  
 a) Iterate through rows `i` from 0 to `n-1`.  
 b) Iterate through columns `j` from 0 to `i`.  
 c) Input the element `lower[i][j]`.  
 d) For columns `j` from `i+1` to `n-1`, set `lower[i][j] = 0`.  
7) Display the lower triangular matrix:  
 a) Iterate through rows `i` from 0 to `n-1`.  
 b) Iterate through columns `j` from 0 to `n-1`.  
 c) Print `lower[i][j]` followed by a space.  
 d) Print a newline character after each row.  
8) Read the elements of the tri-diagonal matrix:  
 a) Iterate through rows `i` from 0 to `n-1`.  
 b) Iterate through columns `j` such that `abs(i - j) <= 1`.  
 c) Input the element `tridiagonal[i][j]`.  
 d) For columns `j` where `abs(i - j) > 1`, set `tridiagonal[i][j] = 0`.  
9) Display the tri-diagonal matrix:  
 a) Iterate through rows `i` from 0 to `n-1`.  
 b) Iterate through columns `j` from 0 to `n-1`.  
 c) Print `tridiagonal[i][j]` followed by a space.  
 d) Print a newline character after each row.  
10) End.

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, i, j;  
  
 printf("\nEnter the size of the matrix: ");  
 scanf("%d", &n);  
  
 int upper[n][n], lower[n][n], tridiagonal[n][n];  
  
 printf("\nEnter elements for the Upper Triangular Matrix:\n");  
 for (i = 0; i < n; i++) {  
 for (j = 0; j < n; j++) {  
 if (j >= i) {  
 scanf("%d", &upper[i][j]);  
 } else {  
 upper[i][j] = 0;  
 }  
 }  
 }  
  
 printf("\nUpper Triangular Matrix:\n");  
 for (i = 0; i < n; i++) {  
 for (j = 0; j < n; j++) {  
 printf("%d ", upper[i][j]);  
 }  
 printf("\n");  
 }  
  
 printf("\nEnter elements for the Lower Triangular Matrix:\n");  
 for (i = 0; i < n; i++) {  
 for (j = 0; j < n; j++) {  
 if (j <= i) {  
 scanf("%d", &lower[i][j]);  
 } else {  
 lower[i][j] = 0;  
 }  
 }  
 }  
  
 printf("\nLower Triangular Matrix:\n");  
 for (i = 0; i < n; i++) {  
 for (j = 0; j < n; j++) {  
 printf("%d ", lower[i][j]);  
 }  
 printf("\n");  
 }  
  
 printf("\nEnter elements for the Tri-diagonal Matrix:\n");  
 for (i = 0; i < n; i++) {  
 for (j = 0; j < n; j++) {  
 if (abs(i - j) <= 1) {  
 scanf("%d", &tridiagonal[i][j]);  
 } else {  
 tridiagonal[i][j] = 0;  
 }  
 }  
 }  
  
 printf("\nTri-diagonal Matrix:\n");  
 for (i = 0; i < n; i++) {  
 for (j = 0; j < n; j++) {  
 printf("%d ", tridiagonal[i][j]);  
 }  
 printf("\n");  
 }  
  
 return 0;  
}

### Output:



## 13. Implement sparse matrices using 3-tuple notation.

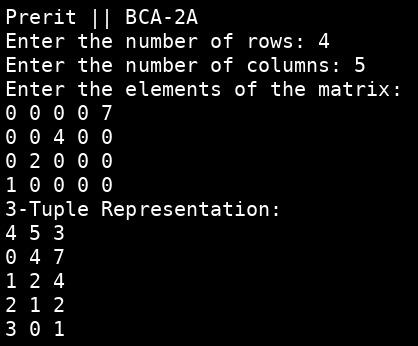
### Algorithm

1) Input the dimensions of the sparse matrix (rows, cols).  
2) Input the elements of the sparse matrix.  
3) Initialize a 3-tuple array (sparse\_matrix) to store the non-zero elements. The first row of this array will store the dimensions and the number of non-zero elements.  
4) Iterate through the original matrix. If an element is non-zero, store its row index, column index, and value into the next available row of the 3-tuple array.  
5) Update the first row of the 3-tuple array with the total number of non-zero elements found.  
6) Print the 3-tuple representation of the sparse matrix.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int rows, cols, i, j, k = 1;  
  
 printf("Enter the number of rows: ");  
 scanf("%d", &rows);  
 printf("Enter the number of columns: ");  
 scanf("%d", &cols);  
  
 int matrix[rows][cols];  
 printf("Enter the elements of the matrix:\n");  
 for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 scanf("%d", &matrix[i][j]);  
 }  
 }  
  
 int sparse\_matrix[rows \* cols][3];  
 sparse\_matrix[0][0] = rows;  
 sparse\_matrix[0][1] = cols;  
 sparse\_matrix[0][2] = 0;  
  
 for (i = 0; i < rows; i++) {  
 for (j = 0; j < cols; j++) {  
 if (matrix[i][j] != 0) {  
 sparse\_matrix[k][0] = i;  
 sparse\_matrix[k][1] = j;  
 sparse\_matrix[k][2] = matrix[i][j];  
 k++;  
 }  
 }  
 }  
  
 sparse\_matrix[0][2] = k - 1;  
  
 printf("3-Tuple Representation:\n");  
 for (i = 0; i < k; i++) {  
 printf("%d %d %d\n", sparse\_matrix[i][0], sparse\_matrix[i][1], sparse\_matrix[i][2]);  
 }  
  
 return 0;  
}

### Output:



## 14. Write a Program to implement Selection Sort.

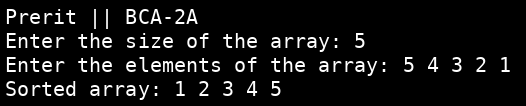
### Algorithm

1) Input the size of the array, n.  
2) Input the array elements.  
3) Iterate from i = 0 to n-2.  
4) Find the index of the minimum element in the subarray arr[i...n-1]. Initialize min\_index = i.  
5) Iterate from j = i+1 to n-1.  
6) If arr[j] < arr[min\_index], then update min\_index = j.  
7) Swap arr[i] with arr[min\_index].  
8) After the outer loop finishes, the array is sorted.  
9) Print the sorted array.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, i, j, min\_index, temp;  
  
 printf("Enter the size of the array: ");  
 scanf("%d", &n);  
  
 int arr[n];  
  
 printf("Enter the elements of the array: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 for (i = 0; i < n - 1; i++) {  
 min\_index = i;  
 for (j = i + 1; j < n; j++) {  
 if (arr[j] < arr[min\_index]) {  
 min\_index = j;  
 }  
 }  
 temp = arr[i];  
 arr[i] = arr[min\_index];  
 arr[min\_index] = temp;  
 }  
  
 printf("Sorted array: ");  
 for (i = 0; i < n; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



## 15. Write a Program to implement Insertion Sort.

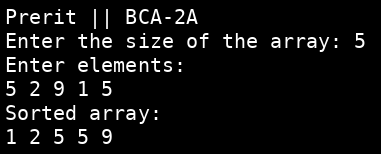
### Algorithm

1) Start.  
2) Input the size of the array, `n`.  
3) Input the elements of the array, `arr[0]` to `arr[n-1]`.  
4) Iterate through the array from the second element (index 1) to the last element (index n-1) using a loop with index `i`.  
5) Inside the loop, store the current element `arr[i]` in a variable `key`.  
6) Initialize another index `j` to `i - 1`.  
7) While `j` is greater than or equal to 0 and `arr[j]` is greater than `key`, do the following:  
 a) Move `arr[j]` to `arr[j+1]` (i.e., `arr[j+1] = arr[j]`).  
 b) Decrement `j` (i.e., `j = j - 1`).  
8) After the inner while loop, insert `key` at `arr[j+1]` (i.e., `arr[j+1] = key`).  
9) After the outer for loop, print the sorted array.  
10) End.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, i, j, key;  
  
 printf("\nEnter the size of the array: ");  
 scanf("%d", &n);  
  
 int arr[n];  
  
 printf("Enter elements:\n");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 for (i = 1; i < n; i++) {  
 key = arr[i];  
 j = i - 1;  
  
 while (j >= 0 && arr[j] > key) {  
 arr[j + 1] = arr[j];  
 j = j - 1;  
 }  
 arr[j + 1] = key;  
 }  
  
 printf("Sorted array:\n");  
 for (i = 0; i < n; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



## 16. Write a Program to implement Bubble Sort.

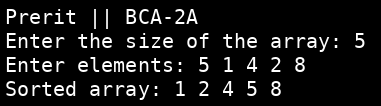
### Algorithm

1) Input the size of the array, n.  
2) Input n elements into the array.  
3) Iterate through the array from i = 0 to n-2.  
4) Iterate through the array from j = 0 to n-i-2.  
5) If array[j] > array[j+1], then swap array[j] and array[j+1].  
6) After completing the outer loop, the array will be sorted.  
7) Print the sorted array.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, i, j, temp;  
  
 printf("Enter the size of the array: ");  
 scanf("%d", &n);  
  
 int arr[n];  
  
 printf("Enter elements: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 for (i = 0; i < n - 1; i++) {  
 for (j = 0; j < n - i - 1; j++) {  
 if (arr[j] > arr[j + 1]) {  
 temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 }  
 }  
 }  
  
 printf("Sorted array: ");  
 for (i = 0; i < n; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



## 17. Write a Program to implement Merge Sort.

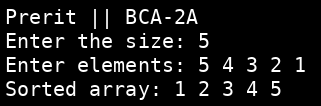
### Algorithm

1) Define a function mergeSort(arr, left, right) that takes an array arr and the left and right indices of the subarray to be sorted.  
2) If left < right, then:  
 a) Calculate the middle index: mid = (left + right) / 2.  
 b) Recursively call mergeSort(arr, left, mid) to sort the left subarray.  
 c) Recursively call mergeSort(arr, mid + 1, right) to sort the right subarray.  
 d) Call the merge(arr, left, mid, right) function to merge the two sorted subarrays.  
3) Define a function merge(arr, left, mid, right) that merges two sorted subarrays into a single sorted subarray.  
4) Calculate the sizes of the two subarrays: n1 = mid - left + 1 and n2 = right - mid.  
5) Create temporary arrays L[n1] and R[n2] to store the two subarrays.  
6) Copy the elements from arr[left...mid] into L[].  
7) Copy the elements from arr[mid+1...right] into R[].  
8) Initialize i = 0, j = 0, and k = left. i and j are the index for left and right array respectively. k is for the main array.  
9) While i < n1 and j < n2:  
 a) If L[i] <= R[j], then set arr[k] = L[i] and increment i.  
 b) Else set arr[k] = R[j] and increment j.  
 c) Increment k.  
10) While i < n1:  
 a) Set arr[k] = L[i] and increment i and k.  
11) While j < n2:  
 a) Set arr[k] = R[j] and increment j and k.  
12) In the main function, get the size of the array from the user.  
13) Get the elements of the array from the user.  
14) Call mergeSort(arr, 0, size - 1) to sort the entire array.  
15) Print the sorted array.

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
void merge(int arr[], int left, int mid, int right) {  
 int n1 = mid - left + 1;  
 int n2 = right - mid;  
 int L[n1], R[n2];  
  
 for (int i = 0; i < n1; i++)  
 L[i] = arr[left + i];  
 for (int j = 0; j < n2; j++)  
 R[j] = arr[mid + 1 + j];  
  
 int i = 0, j = 0, k = left;  
 while (i < n1 && j < n2) {  
 if (L[i] <= R[j]) {  
 arr[k] = L[i];  
 i++;  
 } else {  
 arr[k] = R[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < n1) {  
 arr[k] = L[i];  
 i++;  
 k++;  
 }  
  
 while (j < n2) {  
 arr[k] = R[j];  
 j++;  
 k++;  
 }  
}  
  
void mergeSort(int arr[], int left, int right) {  
 if (left < right) {  
 int mid = left + (right - left) / 2;  
 mergeSort(arr, left, mid);  
 mergeSort(arr, mid + 1, right);  
 merge(arr, left, mid, right);  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int size;  
 printf("\nEnter the size: ");  
 scanf("%d", &size);  
  
 int arr[size];  
 printf("Enter elements: ");  
 for (int i = 0; i < size; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 mergeSort(arr, 0, size - 1);  
  
 printf("Sorted array: ");  
 for (int i = 0; i < size; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



## 18. WAP to implement Singly Linked List that stores data as integer and perform following operations:

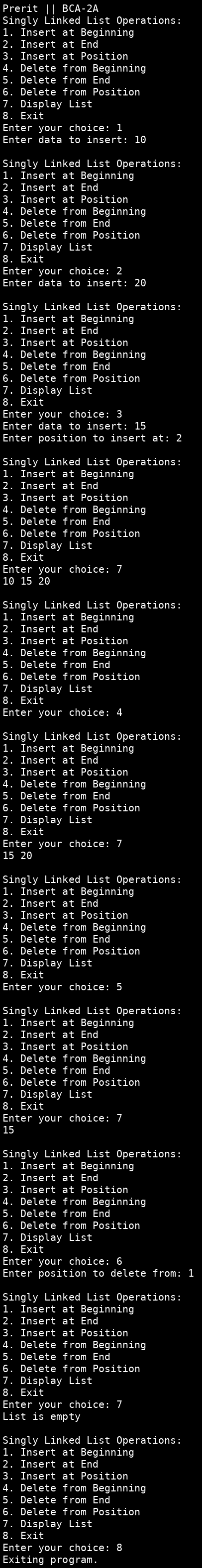
### Algorithm

1) Define a structure Node with integer data and a pointer to the next node.  
2) Initialize the head of the linked list to NULL.  
3) Implement the 'insertAtBeginning' function:  
 a) Create a new node.  
 b) Assign the given data to the new node's data field.  
 c) Set the new node's next pointer to the current head.  
 d) Update the head to point to the new node.  
4) Implement the 'insertAtEnd' function:  
 a) Create a new node.  
 b) Assign the given data to the new node's data field.  
 c) Set the new node's next pointer to NULL.  
 d) If the list is empty (head is NULL), make the new node the head.  
 e) Otherwise, traverse the list to the last node.  
 f) Set the last node's next pointer to the new node.  
5) Implement the 'insertAtPosition' function:  
 a) If position is 1, call insertAtBeginning function.  
 b) Create a new node.  
 c) Assign the given data to the new node's data field.  
 d) Traverse the list to the node just before the desired position.  
 e) Set the new node's next pointer to the current node at the desired position.  
 f) Set the previous node's next pointer to the new node.  
6) Implement the 'deleteFromBeginning' function:  
 a) If the list is empty (head is NULL), return.  
 b) Store the head node in a temporary variable.  
 c) Update the head to point to the next node.  
 d) Free the memory of the original head node.  
7) Implement the 'deleteFromEnd' function:  
 a) If the list is empty (head is NULL), return.  
 b) If the list has only one node, free the head node and set head to NULL.  
 c) Traverse the list to the second-to-last node.  
 d) Store the last node in a temporary variable.  
 e) Set the second-to-last node's next pointer to NULL.  
 f) Free the memory of the original last node.  
8) Implement the 'deleteFromPosition' function:  
 a) If the list is empty (head is NULL), return.  
 b) If the position is 1, call deleteFromBeginning function.  
 c) Traverse the list to the node just before the desired position.  
 d) Store the node to be deleted in a temporary variable.  
 e) Update the previous node's next pointer to skip the node to be deleted.  
 f) Free the memory of the deleted node.  
9) Implement the 'displayList' function:  
 a) If the list is empty (head is NULL), print "List is empty".  
 b) Traverse the list and print the data of each node.  
10) In the main function, present a menu to the user with options for inserting, deleting, and displaying elements.  
11) Based on the user's choice, call the appropriate function.  
12) Repeat step 10 until the user chooses to exit.

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
};  
  
struct Node \*head = NULL;  
  
void insertAtBeginning(int data) {  
 struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = head;  
 head = newNode;  
}  
  
void insertAtEnd(int data) {  
 struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = NULL;  
 if (head == NULL) {  
 head = newNode;  
 return;  
 }  
 struct Node \*temp = head;  
 while (temp->next != NULL) {  
 temp = temp->next;  
 }  
 temp->next = newNode;  
}  
  
void insertAtPosition(int data, int position) {  
 if (position == 1) {  
 insertAtBeginning(data);  
 return;  
 }  
 struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 struct Node \*temp = head;  
 for (int i = 1; i < position - 1 && temp != NULL; i++) {  
 temp = temp->next;  
 }  
 if (temp == NULL) {  
 printf("Invalid position\n");  
 return;  
 }  
 newNode->next = temp->next;  
 temp->next = newNode;  
}  
  
void deleteFromBeginning() {  
 if (head == NULL) return;  
 struct Node \*temp = head;  
 head = head->next;  
 free(temp);  
}  
  
void deleteFromEnd() {  
 if (head == NULL) return;  
 if (head->next == NULL) {  
 free(head);  
 head = NULL;  
 return;  
 }  
 struct Node \*temp = head;  
 while (temp->next->next != NULL) {  
 temp = temp->next;  
 }  
 free(temp->next);  
 temp->next = NULL;  
}  
  
void deleteFromPosition(int position) {  
 if (head == NULL) return;  
 if (position == 1) {  
 deleteFromBeginning();  
 return;  
 }  
 struct Node \*temp = head;  
 for (int i = 1; i < position - 1 && temp != NULL; i++) {  
 temp = temp->next;  
 }  
 if (temp == NULL || temp->next == NULL) {  
 printf("Invalid position\n");  
 return;  
 }  
 struct Node \*nodeToDelete = temp->next;  
 temp->next = temp->next->next;  
 free(nodeToDelete);  
}  
  
void displayList() {  
 if (head == NULL) {  
 printf("List is empty\n");  
 return;  
 }  
 struct Node \*temp = head;  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, data, position;  
 do {  
 printf("\nSingly Linked List Operations:\n");  
 printf("1. Insert at Beginning\n");  
 printf("2. Insert at End\n");  
 printf("3. Insert at Position\n");  
 printf("4. Delete from Beginning\n");  
 printf("5. Delete from End\n");  
 printf("6. Delete from Position\n");  
 printf("7. Display List\n");  
 printf("8. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 insertAtBeginning(data);  
 break;  
 case 2:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 insertAtEnd(data);  
 break;  
 case 3:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 printf("Enter position to insert at: ");  
 scanf("%d", &position);  
 insertAtPosition(data, position);  
 break;  
 case 4:  
 deleteFromBeginning();  
 break;  
 case 5:  
 deleteFromEnd();  
 break;  
 case 6:  
 printf("Enter position to delete from: ");  
 scanf("%d", &position);  
 deleteFromPosition(position);  
 break;  
 case 7:  
 displayList();  
 break;  
 case 8:  
 printf("Exiting program.\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 8);  
 return 0;  
}

### Output:



## 19. Traverse the list to display each element

### Algorithm

1) Initialize a list (array) of elements.  
2) Get the size of the list from the user.  
3) Get each element of the list from the user.  
4) Iterate through the list from the first element to the last element.  
5) For each element in the list, print the element.

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int size, i;  
  
 printf("Enter the size of the list: ");  
 scanf("%d", &size);  
  
 int list[size];  
  
 printf("Enter the elements of the list: ");  
 for (i = 0; i < size; i++) {  
 scanf("%d", &list[i]);  
 }  
  
 printf("Elements of the list are: ");  
 for (i = 0; i < size; i++) {  
 printf("%d ", list[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:

