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

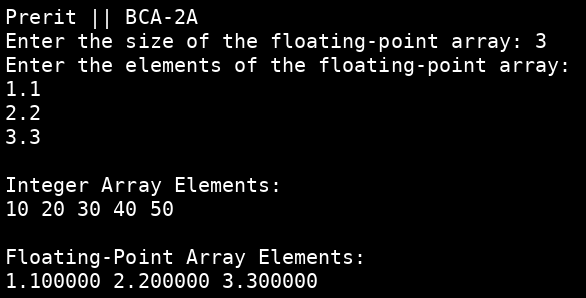
### Algorithm

1. PRINT "Prerit || BCA-2A"  
2. SET `int\_array` = {10, 20, 30, 40, 50}  
3. SET `int\_array\_size` = SIZE\_OF(`int\_array`) / SIZE\_OF(INT)  
4. DECLARE `float\_array[100]` AS FLOAT  
5. READ `float\_array\_size` from user input.  
6. PRINT "Enter the elements of the floating-point array:"  
7. FOR `i` = 0 TO `float\_array\_size` - 1:  
 \* READ `float\_array[i]` from user input  
 [END OF LOOP]  
8. PRINT "\nInteger Array Elements:\n"  
9. FOR `i` = 0 TO `int\_array\_size` - 1:  
 \* PRINT `int\_array[i]` followed by a space.  
 [END OF LOOP]  
10. PRINT "\n"  
11. PRINT "\nFloating-Point Array Elements:\n"  
12. FOR `i` = 0 TO `float\_array\_size` - 1:  
 \* PRINT `float\_array[i]` followed by a space.  
 [END OF LOOP]  
13. PRINT "\n"  
14. EXIT

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int int\_array[] = {10, 20, 30, 40, 50};  
 int int\_array\_size = sizeof(int\_array) / sizeof(int);  
 int float\_array\_size;  
 float float\_array[100];  
  
 printf("\nEnter the size of the floating-point array: ");  
 scanf("%d", &float\_array\_size);  
  
 printf("Enter the elements of the floating-point array:\n");  
 for (int i = 0; i < float\_array\_size; i++) {  
 scanf("%f", &float\_array[i]);  
 }  
  
 printf("\nInteger Array Elements:\n");  
 for (int i = 0; i < int\_array\_size; i++) {  
 printf("%d ", int\_array[i]);  
 }  
 printf("\n");  
  
 printf("\nFloating-Point Array Elements:\n");  
 for (int i = 0; i < float\_array\_size; i++) {  
 printf("%f ", float\_array[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

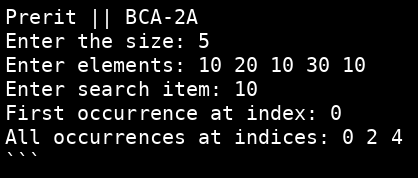
### Algorithm

```  
ALGORITHM FindOccurrences  
  
1. PRINT "Prerit || BCA-2A"  
2. SET firstOccurrence = -1  
3. SET count = 0  
4. INPUT n  
5. INPUT arr[0] ... arr[n-1]  
6. INPUT key  
  
7. FOR i = 0 TO n-1  
 IF arr[i] == key  
 SET firstOccurrence = i  
 EXIT FOR [END OF IF]  
 [END OF LOOP]  
  
8. IF firstOccurrence == -1  
 PRINT "Element not found"  
 ELSE  
 PRINT "First occurrence at index: ", firstOccurrence  
 [END OF IF]  
  
9. FOR i = 0 TO n-1  
 IF arr[i] == key  
 SET allOccurrences[count] = i  
 SET count = count + 1  
 [END OF IF]  
 [END OF LOOP]  
  
10. IF count == 0  
 PRINT "Element not found"  
 ELSE  
 PRINT "All occurrences at indices: "  
 FOR i = 0 TO count - 1  
 PRINT allOccurrences[i], " "  
 [END OF LOOP]  
 PRINT "\n"  
 [END OF IF]  
  
11. EXIT  
```

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, key, firstOccurrence = -1;  
 int arr[100];  
 int allOccurrences[100];  
 int count = 0;  
  
 printf("\nEnter the size: ");  
 scanf("%d", &n);  
  
 printf("Enter elements: ");  
 for (int i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 printf("Enter search item: ");  
 scanf("%d", &key);  
  
 for (int i = 0; i < n; i++) {  
 if (arr[i] == key) {  
 firstOccurrence = i;  
 break;  
 }  
 }  
  
 if (firstOccurrence == -1) {  
 printf("Element not found\n");  
 } else {  
 printf("First occurrence at index: %d\n", firstOccurrence);  
 }  
  
 for (int i = 0; i < n; i++) {  
 if (arr[i] == key) {  
 allOccurrences[count] = i;  
 count++;  
 }  
 }  
  
 if (count == 0) {  
 printf("Element not found\n");  
 } else {  
 printf("All occurrences at indices: ");  
 for (int i = 0; i < count; i++) {  
 printf("%d ", allOccurrences[i]);  
 }  
 printf("\n");  
 }  
  
 return 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

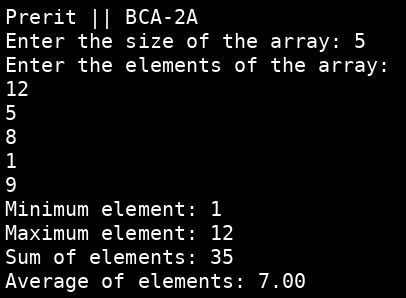
### Algorithm

1. PRINT "Prerit || BCA-2A"  
2. INPUT size  
3. DECLARE arr[size]  
4. PRINT "Enter the elements of the array:"  
5. SET i = 0  
6. WHILE i < size  
 \* INPUT arr[i]  
 \* SET i = i + 1  
[END OF LOOP]  
7. SET min = arr[0]  
8. SET max = arr[0]  
9. SET sum = 0  
10. SET i = 0  
11. WHILE i < size  
 \* IF arr[i] < min  
 \* SET min = arr[i]  
 [END OF IF]  
 \* IF arr[i] > max  
 \* SET max = arr[i]  
 [END OF IF]  
 \* SET sum = sum + arr[i]  
 \* SET i = i + 1  
[END OF LOOP]  
12. SET average = (float)sum / size  
13. PRINT "Minimum element: ", min  
14. PRINT "Maximum element: ", max  
15. PRINT "Sum of elements: ", sum  
16. PRINT "Average of elements: ", average  
17. EXIT

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int size, i;  
  
 printf("Enter the size of the array: ");  
 scanf("%d", &size);  
  
 int arr[size];  
  
 printf("Enter the elements of the array:\n");  
 for (i = 0; i < size; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 int min = arr[0];  
 int max = arr[0];  
 int sum = 0;  
  
 for (i = 0; i < size; i++) {  
 if (arr[i] < min) {  
 min = arr[i];  
 }  
 if (arr[i] > max) {  
 max = arr[i];  
 }  
 sum += arr[i];  
 }  
  
 float average = (float)sum / size;  
  
 printf("Minimum element: %d\n", min);  
 printf("Maximum element: %d\n", max);  
 printf("Sum of elements: %d\n", sum);  
 printf("Average of elements: %.2f\n", average);  
  
 return 0;  
}

### Output:



## 4. Write a program to Merge unsorted arrays

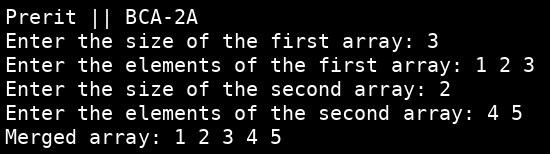
### Algorithm

```  
1. PRINT "Prerit || BCA-2A"  
2. INPUT size1  
3. ALLOCATE memory for arr1 of size size1  
4. FOR i = 0 TO size1 - 1  
 INPUT arr1[i]  
 [END OF LOOP]  
5. INPUT size2  
6. ALLOCATE memory for arr2 of size size2  
7. FOR i = 0 TO size2 - 1  
 INPUT arr2[i]  
 [END OF LOOP]  
8. SET merged\_size = size1 + size2  
9. ALLOCATE memory for merged\_arr of size merged\_size  
10. FOR i = 0 TO size1 - 1  
 SET merged\_arr[i] = arr1[i]  
 [END OF LOOP]  
11. SET i = 0, j = size1  
12. FOR i = 0 TO size2 - 1  
 SET merged\_arr[j] = arr2[i]  
 SET j = j + 1  
 [END OF LOOP]  
13. PRINT "Merged array: "  
14. FOR i = 0 TO merged\_size - 1  
 PRINT merged\_arr[i]  
 [END OF LOOP]  
15. FREE memory allocated for arr1  
16. FREE memory allocated for arr2  
17. FREE memory allocated for merged\_arr  
18. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int size1, size2, i, j, merged\_size;  
 int \*arr1, \*arr2, \*merged\_arr;  
  
 printf("\nEnter the size of the first array: ");  
 scanf("%d", &size1);  
  
 arr1 = (int \*)malloc(size1 \* sizeof(int));  
  
 printf("Enter the elements of the first array: ");  
 for (i = 0; i < size1; i++) {  
 scanf("%d", &arr1[i]);  
 }  
  
 printf("Enter the size of the second array: ");  
 scanf("%d", &size2);  
  
 arr2 = (int \*)malloc(size2 \* sizeof(int));  
  
 printf("Enter the elements of the second array: ");  
 for (i = 0; i < size2; i++) {  
 scanf("%d", &arr2[i]);  
 }  
  
 merged\_size = size1 + size2;  
 merged\_arr = (int \*)malloc(merged\_size \* sizeof(int));  
  
 for (i = 0; i < size1; i++) {  
 merged\_arr[i] = arr1[i];  
 }  
  
 for (i = 0, j = size1; i < size2; i++, j++) {  
 merged\_arr[j] = arr2[i];  
 }  
  
 printf("Merged array: ");  
 for (i = 0; i < merged\_size; i++) {  
 printf("%d ", merged\_arr[i]);  
 }  
 printf("\n");  
  
 free(arr1);  
 free(arr2);  
 free(merged\_arr);  
  
 return 0;  
}

### Output:



## 5. Write a program to Marge sorted arrays

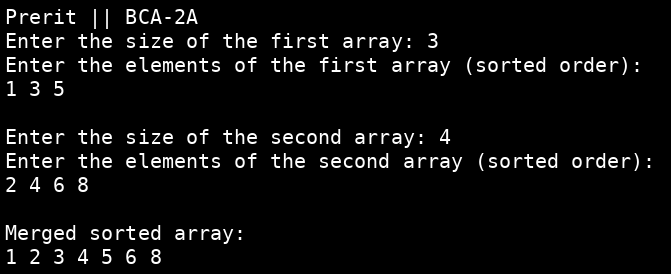
### Algorithm

1. DISPLAY "Prerit || BCA-2A"  
2. INPUT n1  
3. ALLOCATE memory for arr1 of size n1  
4. DISPLAY "Enter the elements of the first array (sorted order):"  
5. FOR i = 0 TO n1 - 1  
 \* INPUT arr1[i]  
 [END OF LOOP]  
6. INPUT n2  
7. ALLOCATE memory for arr2 of size n2  
8. DISPLAY "Enter the elements of the second array (sorted order):"  
9. FOR i = 0 TO n2 - 1  
 \* INPUT arr2[i]  
 [END OF LOOP]  
10. ALLOCATE memory for mergedArray of size n1 + n2  
11. SET i = 0, j = 0, k = 0  
12. WHILE i < n1 AND j < n2  
 \* IF arr1[i] <= arr2[j]  
 \* SET mergedArray[k] = arr1[i]  
 \* SET i = i + 1  
 ELSE  
 \* SET mergedArray[k] = arr2[j]  
 \* SET j = j + 1  
 [END OF IF]  
 \* SET k = k + 1  
 [END OF LOOP]  
13. WHILE i < n1  
 \* SET mergedArray[k] = arr1[i]  
 \* SET i = i + 1  
 \* SET k = k + 1  
 [END OF LOOP]  
14. WHILE j < n2  
 \* SET mergedArray[k] = arr2[j]  
 \* SET j = j + 1  
 \* SET k = k + 1  
 [END OF LOOP]  
15. DISPLAY "Merged sorted array:"  
16. FOR i = 0 TO n1 + n2 - 1  
 \* DISPLAY mergedArray[i]  
 [END OF LOOP]  
17. DISPLAY newline  
18. FREE(arr1)  
19. FREE(arr2)  
20. FREE(mergedArray)  
21. EXIT

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n1, n2, i, j, k;  
  
 printf("\nEnter the size of the first array: ");  
 scanf("%d", &n1);  
  
 int \*arr1 = (int \*)malloc(n1 \* sizeof(int));  
 printf("Enter the elements of the first array (sorted order):\n");  
 for (i = 0; i < n1; i++) {  
 scanf("%d", &arr1[i]);  
 }  
  
 printf("\nEnter the size of the second array: ");  
 scanf("%d", &n2);  
  
 int \*arr2 = (int \*)malloc(n2 \* sizeof(int));  
 printf("Enter the elements of the second array (sorted order):\n");  
 for (i = 0; i < n2; i++) {  
 scanf("%d", &arr2[i]);  
 }  
  
 int \*mergedArray = (int \*)malloc((n1 + n2) \* sizeof(int));  
  
 i = 0;  
 j = 0;  
 k = 0;  
  
 while (i < n1 && j < n2) {  
 if (arr1[i] <= arr2[j]) {  
 mergedArray[k] = arr1[i];  
 i++;  
 } else {  
 mergedArray[k] = arr2[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < n1) {  
 mergedArray[k] = arr1[i];  
 i++;  
 k++;  
 }  
  
 while (j < n2) {  
 mergedArray[k] = arr2[j];  
 j++;  
 k++;  
 }  
  
 printf("\nMerged sorted array:\n");  
 for (i = 0; i < n1 + n2; i++) {  
 printf("%d ", mergedArray[i]);  
 }  
 printf("\n");  
  
 free(arr1);  
 free(arr2);  
 free(mergedArray);  
  
 return 0;  
}

### Output:



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

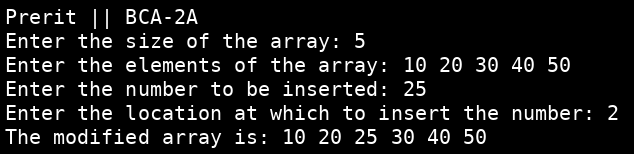
### Algorithm

\*\*Algorithm: Array Insertion\*\*  
  
Step 1: PRINT "Prerit || BCA-2A"  
Step 2: INPUT size  
Step 3: INPUT arr[0] to arr[size-1]  
Step 4: INPUT num  
Step 5: INPUT location  
Step 6: IF location < 0 OR location > size THEN  
 Step 7: PRINT "Invalid location. Location should be between 0 and size"  
 Step 8: EXIT  
 [END OF IF]  
Step 9: SET i = size - 1  
Step 10: WHILE i >= location DO  
 Step 11: arr[i + 1] = arr[i]  
 Step 12: SET i = i - 1  
 [END OF LOOP]  
Step 13: arr[location] = num  
Step 14: SET size = size + 1  
Step 15: SET i = 0  
Step 16: WHILE i < size DO  
 Step 17: PRINT arr[i]  
 Step 18: SET i = i + 1  
 [END OF LOOP]  
Step 19: EXIT

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[100];  
 int 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);  
  
 if (location < 0 || location > size) {  
 printf("Invalid location. Location should be between 0 and %d\n", size);  
 return 1;  
 }  
  
 for (i = size - 1; i >= location; i--) {  
 arr[i + 1] = arr[i];  
 }  
  
 arr[location] = num;  
 size++;  
  
 printf("The modified 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.

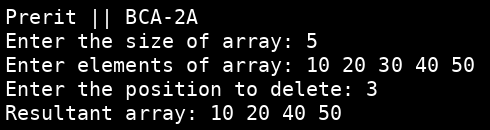
### Algorithm

1. PRINT "Prerit || BCA-2A"  
2. DECLARE `arr` as an array of 100 integers.  
3. DECLARE `n`, `position`, `i` as integers.  
4. PRINT "Enter the size of array: "  
5. READ `n`  
6. PRINT "Enter elements of array: "  
7. SET `i` = 0  
8. WHILE `i` < `n`  
 \* READ `arr[i]`  
 \* SET `i` = `i` + 1  
 [END OF LOOP]  
9. PRINT "Enter the position to delete: "  
10. READ `position`  
11. IF `position` < 1 OR `position` > `n`  
 \* PRINT "Invalid position\n"  
 ELSE  
 \* SET `i` = `position` - 1  
 \* WHILE `i` < `n` - 1  
 \* SET `arr[i]` = `arr[i + 1]`  
 \* SET `i` = `i` + 1  
 [END OF LOOP]  
 \* SET `n` = `n` - 1  
 \* PRINT "Resultant array: "  
 \* SET `i` = 0  
 \* WHILE `i` < `n`  
 \* PRINT `arr[i]`  
 \* PRINT " "  
 \* SET `i` = `i` + 1  
 [END OF LOOP]  
 \* PRINT "\n"  
 [END OF IF]  
12. EXIT

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[100], n, position, i;  
  
 printf("Enter the size of array: ");  
 scanf("%d", &n);  
  
 printf("Enter elements of array: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 printf("Enter the position to delete: ");  
 scanf("%d", &position);  
  
 if (position < 1 || position > n) {  
 printf("Invalid position\n");  
 } else {  
 for (i = position - 1; i < n - 1; i++) {  
 arr[i] = arr[i + 1];  
 }  
  
 n--;  
  
 printf("Resultant array: ");  
 for (i = 0; i < n; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
 }  
  
 return 0;  
}

### Output:



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

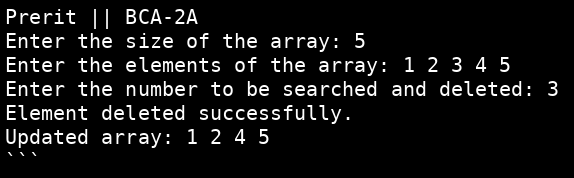
### Algorithm

\*\*Algorithm: Array Element Search and Deletion\*\*  
  
1. SET FOUND = 0  
2. INPUT array size, N  
3. INPUT array elements, ARR[0] to ARR[N-1]  
4. INPUT search key, KEY  
5. FOR I = 0 to N-1  
 \* IF ARR[I] == KEY THEN  
 \* SET FOUND = 1  
 \* FOR J = I to N-2  
 \* ARR[J] = ARR[J+1]  
 [END OF LOOP]  
 \* N = N - 1  
 \* BREAK  
 [END OF IF]  
 [END OF LOOP]  
6. IF FOUND == 1 THEN  
 \* OUTPUT "Element deleted successfully."  
 \* OUTPUT "Updated array: "  
 \* FOR I = 0 to N-1  
 \* OUTPUT ARR[I]  
 [END OF LOOP]  
 [END OF IF]  
7. ELSE  
 \* OUTPUT "Element not found."  
 [END OF ELSE]  
8. EXIT

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[100], n, i, key, found = 0, j;  
  
 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 number to be searched and 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

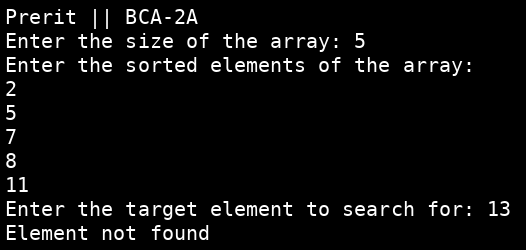
### Algorithm

```  
Algorithm BinarySearchMain  
  
Step 1: PRINT "Prerit || BCA-2A"  
Step 2: PRINT "Enter the size of the array: "  
Step 3: READ size  
Step 4: ALLOCATE memory for array arr of size 'size'  
Step 5: PRINT "Enter the sorted elements of the array:"  
Step 6: FOR i = 0 to size - 1 DO  
 READ arr[i]  
[END OF LOOP]  
Step 7: PRINT "Enter the target element to search for: "  
Step 8: READ target  
Step 9: SET index = BinarySearch(arr, target, 0, size - 1)  
Step 10: IF index == -1 THEN  
 PRINT "Element not found"  
ELSE  
 PRINT "Element found at index: ", index  
[END OF IF]  
Step 11: FREE arr  
Step 12: EXIT  
  
Algorithm BinarySearch(arr, target, low, high)  
  
Step 1: WHILE low <= high DO  
 SET mid = (low + high) / 2  
 IF arr[mid] == target THEN  
 RETURN mid  
 ELSE IF arr[mid] < target THEN  
 SET low = mid + 1  
 ELSE  
 SET high = mid - 1  
 [END OF IF]  
[END OF LOOP]  
Step 2: RETURN -1  
```

### Code:

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

### Output:



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

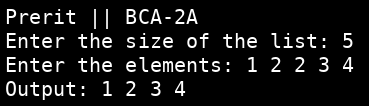
### Algorithm

```  
ALGORITHM RemoveDuplicates  
  
1. DISPLAY "Prerit || BCA-2A"  
2. INPUT n  
3. SET list AS an array of size n  
4. SET uniqueList AS an array of size n  
5. SET uniqueSize = 0  
6. DISPLAY "Enter the elements: "  
7. FOR i = 0 to n-1  
 INPUT list[i]  
 [END OF LOOP]  
8. FOR i = 0 to n-1  
 SET isDuplicate = 0  
 FOR j = 0 to uniqueSize-1  
 IF list[i] == uniqueList[j] THEN  
 SET isDuplicate = 1  
 BREAK  
 [END OF IF]  
 [END OF LOOP]  
 IF isDuplicate == 0 THEN  
 SET uniqueList[uniqueSize] = list[i]  
 SET uniqueSize = uniqueSize + 1  
 [END OF IF]  
 [END OF LOOP]  
9. DISPLAY "Output: "  
10. FOR i = 0 to uniqueSize-1  
 DISPLAY uniqueList[i]  
 [END OF LOOP]  
11. DISPLAY newline character  
12. EXIT  
```

### Code:

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

### Output:



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

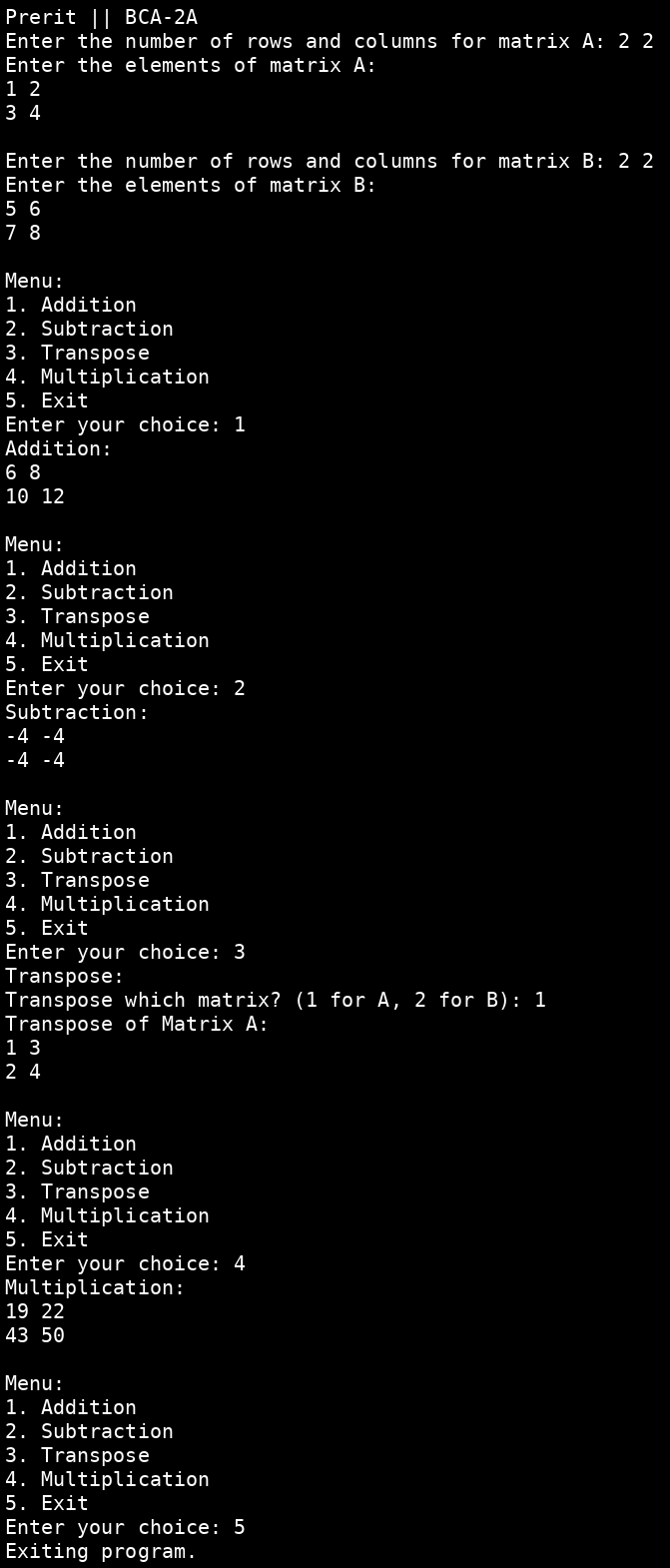
### Algorithm

```  
ALGORITHM MatrixOperations  
  
// Initialization  
1. PRINT "Prerit || BCA-2A"  
2. DECLARE rowsA, colsA, rowsB, colsB, choice, i, j, k as INTEGER  
3. DECLARE A[10][10], B[10][10], C[10][10] as INTEGER  
  
// Input Matrices  
4. PRINT "Enter the number of rows and columns for matrix A: "  
5. READ rowsA, colsA  
6. PRINT "Enter the elements of matrix A:"  
7. FOR i = 0 TO rowsA - 1 DO  
 FOR j = 0 TO colsA - 1 DO  
 READ A[i][j]  
 [END OF LOOP]  
 [END OF LOOP]  
8. PRINT "Enter the number of rows and columns for matrix B: "  
9. READ rowsB, colsB  
10. PRINT "Enter the elements of matrix B:"  
11. FOR i = 0 TO rowsB - 1 DO  
 FOR j = 0 TO colsB - 1 DO  
 READ B[i][j]  
 [END OF LOOP]  
 [END OF LOOP]  
  
// Menu Loop  
12. DO  
13. PRINT "\nMenu:\n1. Addition\n2. Subtraction\n3. Transpose\n4. Multiplication\n5. Exit\nEnter your choice: "  
14. READ choice  
  
 // Switch statement for menu choices  
15. SWITCH choice  
 // Case 1: Addition  
16. CASE 1:  
17. IF rowsA != rowsB OR colsA != colsB THEN  
18. PRINT "Matrices cannot be added.\n"  
19. ELSE  
20. PRINT "Addition:\n"  
21. FOR i = 0 TO rowsA - 1 DO  
22. FOR j = 0 TO colsA - 1 DO  
23. SET C[i][j] = A[i][j] + B[i][j]  
24. PRINT C[i][j], " "  
25. [END OF LOOP]  
26. PRINT "\n"  
27. [END OF LOOP]  
28. [END OF IF]  
  
 // Case 2: Subtraction  
29. CASE 2:  
30. IF rowsA != rowsB OR colsA != colsB THEN  
31. PRINT "Matrices cannot be subtracted.\n"  
32. ELSE  
33. PRINT "Subtraction:\n"  
34. FOR i = 0 TO rowsA - 1 DO  
35. FOR j = 0 TO colsA - 1 DO  
36. SET C[i][j] = A[i][j] - B[i][j]  
37. PRINT C[i][j], " "  
38. [END OF LOOP]  
39. PRINT "\n"  
40. [END OF LOOP]  
41. [END OF IF]  
  
 // Case 3: Transpose  
42. CASE 3:  
43. PRINT "Transpose:\n"  
44. PRINT "Transpose which matrix? (1 for A, 2 for B): "  
45. READ matrix\_choice  
46. IF matrix\_choice == 1 THEN  
47. PRINT "Transpose of Matrix A:\n"  
48. FOR i = 0 TO colsA - 1 DO  
49. FOR j = 0 TO rowsA - 1 DO  
50. PRINT A[j][i], " "  
51. [END OF LOOP]  
52. PRINT "\n"  
53. [END OF LOOP]  
54. ELSE IF matrix\_choice == 2 THEN  
55. PRINT "Transpose of Matrix B:\n"  
56. FOR i = 0 TO colsB - 1 DO  
57. FOR j = 0 TO rowsB - 1 DO  
58. PRINT B[j][i], " "  
59. [END OF LOOP]  
60. PRINT "\n"  
61. [END OF LOOP]  
62. ELSE  
63. PRINT "Invalid Matrix Choice\n"  
64. [END OF IF]  
  
 // Case 4: Multiplication  
65. CASE 4:  
66. IF colsA != rowsB THEN  
67. PRINT "Matrices cannot be multiplied.\n"  
68. ELSE  
69. PRINT "Multiplication:\n"  
70. FOR i = 0 TO rowsA - 1 DO  
71. FOR j = 0 TO colsB - 1 DO  
72. SET C[i][j] = 0  
73. FOR k = 0 TO colsA - 1 DO  
74. SET C[i][j] = C[i][j] + A[i][k] \* B[k][j]  
75. [END OF LOOP]  
76. PRINT C[i][j], " "  
77. [END OF LOOP]  
78. PRINT "\n"  
79. [END OF LOOP]  
80. [END OF IF]  
  
 // Case 5: Exit  
81. CASE 5:  
82. PRINT "Exiting program.\n"  
  
 // Default Case  
83. DEFAULT:  
84. PRINT "Invalid choice. Please try again.\n"  
85. END SWITCH  
  
86. UNTIL choice == 5  
  
// Termination  
87. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int rowsA, colsA, rowsB, colsB, choice, i, j, k;  
 int A[10][10], B[10][10], C[10][10];  
  
 printf("\nEnter 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", &A[i][j]);  
 }  
 }  
  
 printf("\nEnter 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", &B[i][j]);  
 }  
 }  
  
 do {  
 printf("\nMenu:\n");  
 printf("1. Addition\n");  
 printf("2. Subtraction\n");  
 printf("3. Transpose\n");  
 printf("4. Multiplication\n");  
 printf("5. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 if (rowsA != rowsB || colsA != colsB) {  
 printf("Matrices cannot be added.\n");  
 } else {  
 printf("Addition:\n");  
 for (i = 0; i < rowsA; i++) {  
 for (j = 0; j < colsA; j++) {  
 C[i][j] = A[i][j] + B[i][j];  
 printf("%d ", C[i][j]);  
 }  
 printf("\n");  
 }  
 }  
 break;  
  
 case 2:  
 if (rowsA != rowsB || colsA != colsB) {  
 printf("Matrices cannot be subtracted.\n");  
 } else {  
 printf("Subtraction:\n");  
 for (i = 0; i < rowsA; i++) {  
 for (j = 0; j < colsA; j++) {  
 C[i][j] = A[i][j] - B[i][j];  
 printf("%d ", C[i][j]);  
 }  
 printf("\n");  
 }  
 }  
 break;  
  
 case 3:  
 printf("Transpose:\n");  
 int matrix\_choice;  
 printf("Transpose which matrix? (1 for A, 2 for B): ");  
 scanf("%d", &matrix\_choice);  
  
 if (matrix\_choice == 1) {  
 printf("Transpose of Matrix A:\n");  
 for (i = 0; i < colsA; i++) {  
 for (j = 0; j < rowsA; j++) {  
 printf("%d ", A[j][i]);  
 }  
 printf("\n");  
 }  
 } else if (matrix\_choice == 2) {  
 printf("Transpose of Matrix B:\n");  
 for (i = 0; i < colsB; i++) {  
 for (j = 0; j < rowsB; j++) {  
 printf("%d ", B[j][i]);  
 }  
 printf("\n");  
 }  
 } else {  
 printf("Invalid Matrix Choice\n");  
 }  
 break;  
  
 case 4:  
 if (colsA != rowsB) {  
 printf("Matrices cannot be multiplied.\n");  
 } else {  
 printf("Multiplication:\n");  
 for (i = 0; i < rowsA; i++) {  
 for (j = 0; j < colsB; j++) {  
 C[i][j] = 0;  
 for (k = 0; k < colsA; k++) {  
 C[i][j] += A[i][k] \* B[k][j];  
 }  
 printf("%d ", C[i][j]);  
 }  
 printf("\n");  
 }  
 }  
 break;  
  
 case 5:  
 printf("Exiting program.\n");  
 break;  
  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 5);  
  
 return 0;  
}

### Output:



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

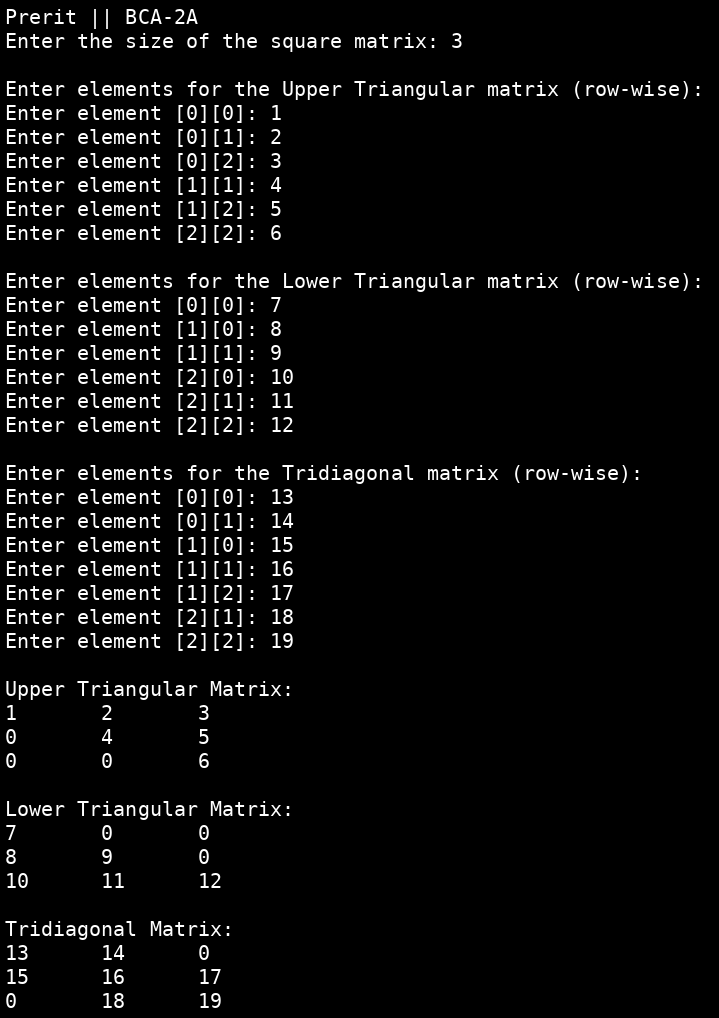
### Algorithm

```  
ALGORITHM MatrixOperations  
  
Step 1: PRINT "Prerit || BCA-2A"  
  
Step 2: PRINT "Enter the size of the square matrix: "  
  
Step 3: READ n  
  
Step 4: ALLOCATE upper[n][n], lower[n][n], tridiagonal[n][n]  
  
Step 5: PRINT "Enter elements for the Upper Triangular matrix (row-wise):"  
  
Step 6: FOR i = 0 TO n-1  
 FOR j = i TO n-1  
 PRINT "Enter element [" i "][" j "]: "  
 READ upper[i][j]  
 [END OF LOOP]  
[END OF LOOP]  
  
Step 7: PRINT "Enter elements for the Lower Triangular matrix (row-wise):"  
  
Step 8: FOR i = 0 TO n-1  
 FOR j = 0 TO i  
 PRINT "Enter element [" i "][" j "]: "  
 READ lower[i][j]  
 [END OF LOOP]  
[END OF LOOP]  
  
Step 9: PRINT "Enter elements for the Tridiagonal matrix (row-wise):"  
  
Step 10: FOR i = 0 TO n-1  
 FOR j = 0 TO n-1  
 IF i == j OR i == j - 1 OR i == j + 1  
 PRINT "Enter element [" i "][" j "]: "  
 READ tridiagonal[i][j]  
 [END OF IF]  
 [END OF LOOP]  
[END OF LOOP]  
  
Step 11: PRINT "Upper Triangular Matrix:"  
  
Step 12: FOR i = 0 TO n-1  
 FOR j = 0 TO n-1  
 IF i <= j  
 PRINT upper[i][j] "\t"  
 ELSE  
 PRINT "0\t"  
 [END OF IF]  
 [END OF LOOP]  
 PRINT "\n"  
[END OF LOOP]  
  
Step 13: PRINT "Lower Triangular Matrix:"  
  
Step 14: FOR i = 0 TO n-1  
 FOR j = 0 TO n-1  
 IF i >= j  
 PRINT lower[i][j] "\t"  
 ELSE  
 PRINT "0\t"  
 [END OF IF]  
 [END OF LOOP]  
 PRINT "\n"  
[END OF LOOP]  
  
Step 15: PRINT "Tridiagonal Matrix:"  
  
Step 16: FOR i = 0 TO n-1  
 FOR j = 0 TO n-1  
 IF i == j OR i == j - 1 OR i == j + 1  
 PRINT tridiagonal[i][j] "\t"  
 ELSE  
 PRINT "0\t"  
 [END OF IF]  
 [END OF LOOP]  
 PRINT "\n"  
[END OF LOOP]  
  
Step 17: EXIT  
```

### Code:

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

### Output:



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

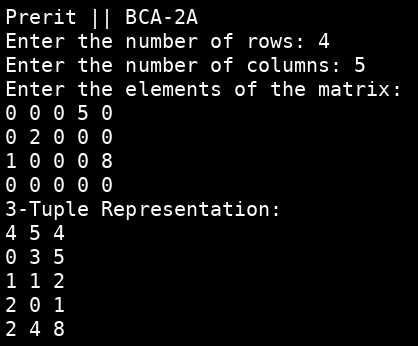
### Algorithm

```  
// Algorithm to convert a sparse matrix to a 3-tuple representation  
  
Step 1: PRINT "Prerit || BCA-2A"  
  
Step 2: READ rows  
  
Step 3: READ cols  
  
Step 4: SET sparseMatrix[rows][cols]  
  
Step 5: SET i = 0  
  
Step 6: SET count = 0  
  
Step 7: WHILE i < rows  
 SET j = 0  
 WHILE j < cols  
 READ sparseMatrix[i][j]  
 IF sparseMatrix[i][j] != 0  
 SET count = count + 1  
 [END OF IF]  
 SET j = j + 1  
 [END OF LOOP]  
 SET i = i + 1  
[END OF LOOP]  
  
Step 8: SET tupleMatrix[count + 1][3]  
  
Step 9: SET tupleMatrix[0][0] = rows  
  
Step 10: SET tupleMatrix[0][1] = cols  
  
Step 11: SET tupleMatrix[0][2] = count  
  
Step 12: SET i = 0  
  
Step 13: SET k = 1  
  
Step 14: WHILE i < rows  
 SET j = 0  
 WHILE j < cols  
 IF sparseMatrix[i][j] != 0  
 SET tupleMatrix[k][0] = i  
 SET tupleMatrix[k][1] = j  
 SET tupleMatrix[k][2] = sparseMatrix[i][j]  
 SET k = k + 1  
 [END OF IF]  
 SET j = j + 1  
 [END OF LOOP]  
 SET i = i + 1  
[END OF LOOP]  
  
Step 15: PRINT "3-Tuple Representation:"  
  
Step 16: SET i = 0  
  
Step 17: WHILE i <= count  
 PRINT tupleMatrix[i][0], tupleMatrix[i][1], tupleMatrix[i][2]  
 SET i = i + 1  
[END OF LOOP]  
  
Step 18: EXIT  
```

### Code:

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

### Output:



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

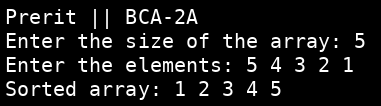
### Algorithm

Here's the algorithm for the Selection Sort function, followed by the main function's algorithm.  
  
\*\*Selection Sort Algorithm (selectionSort(arr, n))\*\*  
  
INPUT: Array `arr` of size `n`.  
  
Step 1: SET `i` = 0.  
Step 2: WHILE `i` < `n` - 1, DO  
 Step 3: SET `min\_index` = `i`.  
 Step 4: SET `j` = `i` + 1.  
 Step 5: WHILE `j` < `n`, DO  
 Step 6: IF `arr[j]` < `arr[min\_index]`, THEN  
 Step 7: SET `min\_index` = `j`.  
 [END OF IF]  
 Step 8: SET `j` = `j` + 1.  
 [END OF LOOP]  
 Step 9: IF `min\_index` != `i`, THEN  
 Step 10: SET `temp` = `arr[i]`.  
 Step 11: SET `arr[i]` = `arr[min\_index]`.  
 Step 12: SET `arr[min\_index]` = `temp`.  
 [END OF IF]  
 Step 13: SET `i` = `i` + 1.  
[END OF LOOP]  
Step 14: EXIT.  
  
\*\*Main Function Algorithm (main())\*\*  
  
Step 1: PRINT "Prerit || BCA-2A".  
Step 2: DECLARE `arr` as an array of size 100.  
Step 3: DECLARE `n`, `i` as integers.  
Step 4: PRINT "Enter the size of the array: ".  
Step 5: READ `n`.  
Step 6: PRINT "Enter the elements: ".  
Step 7: SET `i` = 0.  
Step 8: WHILE `i` < `n`, DO  
 Step 9: READ `arr[i]`.  
 Step 10: SET `i` = `i` + 1.  
 [END OF LOOP]  
Step 11: CALL `selectionSort(arr, n)`.  
Step 12: PRINT "Sorted array: ".  
Step 13: SET `i` = 0.  
Step 14: WHILE `i` < `n`, DO  
 Step 15: PRINT `arr[i]` followed by a space.  
 Step 16: SET `i` = `i` + 1.  
 [END OF LOOP]  
Step 17: PRINT a newline character.  
Step 18: EXIT.

### Code:

#include <stdio.h>  
  
void selectionSort(int arr[], int n) {  
 int i, j, min\_index, temp;  
  
 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;  
 }  
 }  
 if (min\_index != i) {  
 temp = arr[i];  
 arr[i] = arr[min\_index];  
 arr[min\_index] = temp;  
 }  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int arr[100], n, i;  
  
 printf("\nEnter the size of the array: ");  
 scanf("%d", &n);  
  
 printf("Enter the elements: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &arr[i]);  
 }  
  
 selectionSort(arr, n);  
  
 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.

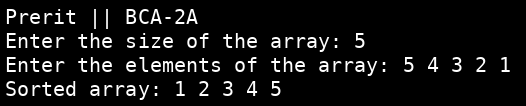
### Algorithm

1. START  
2. DISPLAY "Prerit || BCA-2A"  
3. DISPLAY "Enter the size of the array: "  
4. READ n  
5. DECLARE array arr of size n  
6. DISPLAY "Enter the elements of the array: "  
7. SET i = 0  
8. WHILE i < n  
 \* READ arr[i]  
 \* SET i = i + 1  
 [END OF LOOP]  
9. SET i = 1  
10. WHILE i < n  
 \* SET key = arr[i]  
 \* SET j = i - 1  
 \* WHILE j >= 0 AND arr[j] > key  
 \* SET arr[j + 1] = arr[j]  
 \* SET j = j - 1  
 [END OF LOOP]  
 \* SET arr[j + 1] = key  
 \* SET i = i + 1  
 [END OF LOOP]  
11. DISPLAY "Sorted array: "  
12. SET i = 0  
13. WHILE i < n  
 \* DISPLAY arr[i] followed by a space  
 \* SET i = i + 1  
 [END OF LOOP]  
14. DISPLAY newline  
15. EXIT

### 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 the elements of the array: ");  
 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: ");  
 for (i = 0; i < n; i++) {  
 printf("%d ", arr[i]);  
 }  
 printf("\n");  
  
 return 0;  
}

### Output:



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

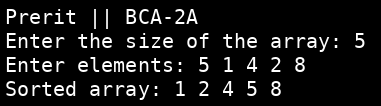
### Algorithm

1. PRINT "Prerit || BCA-2A"  
2. INPUT n (array size)  
3. DECLARE arr[n] (integer array)  
4. PRINT "Enter elements:"  
5. FOR i = 0 TO n-1 DO  
 \* INPUT arr[i]  
 [END OF LOOP]  
6. FOR i = 0 TO n-2 DO  
 \* FOR j = 0 TO n-i-2 DO  
 \* IF arr[j] > arr[j+1] THEN  
 \* SET temp = arr[j]  
 \* SET arr[j] = arr[j+1]  
 \* SET arr[j+1] = temp  
 [END OF IF]  
 [END OF LOOP]  
 [END OF LOOP]  
7. PRINT "Sorted array: "  
8. FOR i = 0 TO n-1 DO  
 \* PRINT arr[i] followed by a space.  
 [END OF LOOP]  
9. PRINT newline character.  
10. EXIT

### 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.

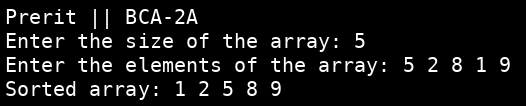
### Algorithm

```  
MERGE SORT ALGORITHM:  
  
1. START  
2. PRINT "Prerit || BCA-2A"  
3. INPUT SIZE  
4. DECLARE ARRAY ARR[SIZE]  
5. FOR I = 0 TO SIZE - 1  
 INPUT ARR[I]  
 [END OF LOOP]  
6. CALL MERGESORT(ARR, 0, SIZE - 1)  
7. PRINT "Sorted array: "  
8. FOR I = 0 TO SIZE - 1  
 PRINT ARR[I] + " "  
 [END OF LOOP]  
9. PRINT NEWLINE  
10. EXIT  
  
MERGESORT(ARR, LEFT, RIGHT):  
  
1. IF LEFT < RIGHT THEN  
 SET MID = LEFT + (RIGHT - LEFT) / 2  
 CALL MERGESORT(ARR, LEFT, MID)  
 CALL MERGESORT(ARR, MID + 1, RIGHT)  
 CALL MERGE(ARR, LEFT, MID, RIGHT)  
 [END OF IF]  
2. EXIT  
  
MERGE(ARR, LEFT, MID, RIGHT):  
  
1. SET N1 = MID - LEFT + 1  
2. SET N2 = RIGHT - MID  
3. DECLARE ARRAY L[N1]  
4. DECLARE ARRAY R[N2]  
5. FOR I = 0 TO N1 - 1  
 SET L[I] = ARR[LEFT + I]  
 [END OF LOOP]  
6. FOR J = 0 TO N2 - 1  
 SET R[J] = ARR[MID + 1 + J]  
 [END OF LOOP]  
7. SET I = 0  
8. SET J = 0  
9. SET K = LEFT  
10. WHILE I < N1 AND J < N2  
 IF L[I] <= R[J] THEN  
 SET ARR[K] = L[I]  
 INCREMENT I  
 ELSE  
 SET ARR[K] = R[J]  
 INCREMENT J  
 [END OF IF]  
 INCREMENT K  
 [END OF LOOP]  
11. WHILE I < N1  
 SET ARR[K] = L[I]  
 INCREMENT I  
 INCREMENT K  
 [END OF LOOP]  
12. WHILE J < N2  
 SET ARR[K] = R[J]  
 INCREMENT J  
 INCREMENT K  
 [END OF LOOP]  
13. EXIT  
```

### 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 of the array: ");  
 scanf("%d", &size);  
 int arr[size];  
 printf("Enter the elements of the array: ");  
 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:

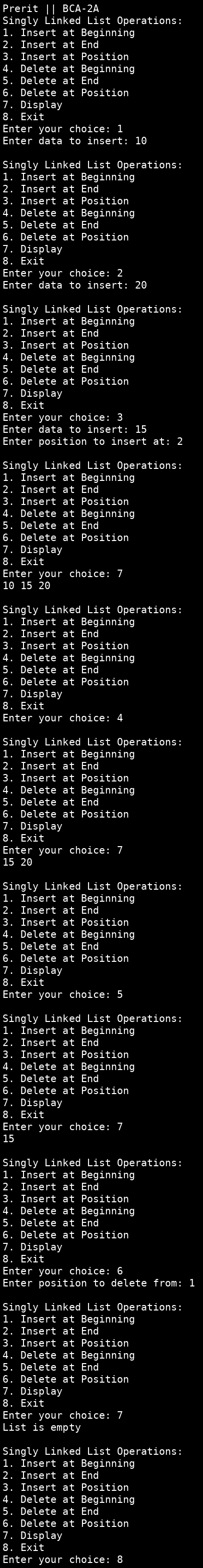
### Algorithm

```  
// Algorithm for Singly Linked List Operations  
  
// Initialization:  
// SET HEAD = NULL  
  
// Function: insertAtBeginning(DATA)  
1. ALLOCATE NEWNODE  
2. SET NEWNODE→DATA = DATA  
3. SET NEWNODE→NEXT = HEAD  
4. SET HEAD = NEWNODE  
 [END OF FUNCTION]  
  
// Function: insertAtEnd(DATA)  
1. ALLOCATE NEWNODE  
2. SET NEWNODE→DATA = DATA  
3. SET NEWNODE→NEXT = NULL  
4. IF HEAD == NULL THEN  
5. SET HEAD = NEWNODE  
6. EXIT FUNCTION  
 [END OF IF]  
7. SET TEMP = HEAD  
8. WHILE TEMP→NEXT != NULL DO  
9. SET TEMP = TEMP→NEXT  
 [END OF LOOP]  
10. SET TEMP→NEXT = NEWNODE  
 [END OF FUNCTION]  
  
// Function: insertAtPosition(DATA, POSITION)  
1. IF POSITION == 1 THEN  
2. CALL insertAtBeginning(DATA)  
3. EXIT FUNCTION  
 [END OF IF]  
4. ALLOCATE NEWNODE  
5. SET NEWNODE→DATA = DATA  
6. SET TEMP = HEAD  
7. SET I = 1  
8. WHILE I < POSITION - 1 AND TEMP != NULL DO  
9. SET TEMP = TEMP→NEXT  
10. SET I = I + 1  
 [END OF LOOP]  
11. IF TEMP == NULL THEN  
12. PRINT "Invalid position"  
13. EXIT FUNCTION  
 [END OF IF]  
14. SET NEWNODE→NEXT = TEMP→NEXT  
15. SET TEMP→NEXT = NEWNODE  
 [END OF FUNCTION]  
  
// Function: deleteAtBeginning()  
1. IF HEAD == NULL THEN  
2. PRINT "List is empty"  
3. EXIT FUNCTION  
 [END OF IF]  
4. SET TEMP = HEAD  
5. SET HEAD = HEAD→NEXT  
6. FREE TEMP  
 [END OF FUNCTION]  
  
// Function: deleteAtEnd()  
1. IF HEAD == NULL THEN  
2. PRINT "List is empty"  
3. EXIT FUNCTION  
 [END OF IF]  
4. IF HEAD→NEXT == NULL THEN  
5. FREE HEAD  
6. SET HEAD = NULL  
7. EXIT FUNCTION  
 [END OF IF]  
8. SET TEMP = HEAD  
9. WHILE TEMP→NEXT→NEXT != NULL DO  
10. SET TEMP = TEMP→NEXT  
 [END OF LOOP]  
11. FREE TEMP→NEXT  
12. SET TEMP→NEXT = NULL  
 [END OF FUNCTION]  
  
// Function: deleteAtPosition(POSITION)  
1. IF HEAD == NULL THEN  
2. PRINT "List is empty"  
3. EXIT FUNCTION  
 [END OF IF]  
4. IF POSITION == 1 THEN  
5. CALL deleteAtBeginning()  
6. EXIT FUNCTION  
 [END OF IF]  
7. SET TEMP = HEAD  
8. SET I = 1  
9. WHILE I < POSITION - 1 AND TEMP != NULL DO  
10. SET TEMP = TEMP→NEXT  
11. SET I = I + 1  
 [END OF LOOP]  
12. IF TEMP == NULL OR TEMP→NEXT == NULL THEN  
13. PRINT "Invalid position"  
14. EXIT FUNCTION  
 [END OF IF]  
15. SET NODETODELETE = TEMP→NEXT  
16. SET TEMP→NEXT = TEMP→NEXT→NEXT  
17. FREE NODETODELETE  
 [END OF FUNCTION]  
  
// Function: display()  
1. IF HEAD == NULL THEN  
2. PRINT "List is empty"  
3. EXIT FUNCTION  
 [END OF IF]  
4. SET TEMP = HEAD  
5. WHILE TEMP != NULL DO  
6. PRINT TEMP→DATA  
7. SET TEMP = TEMP→NEXT  
 [END OF LOOP]  
 [END OF FUNCTION]  
  
// Main Function  
1. PRINT "Singly Linked List Operations:"  
2. GET CHOICE  
3. SWITCH (CHOICE)  
4. CASE 1: GET DATA, CALL insertAtBeginning(DATA)  
5. CASE 2: GET DATA, CALL insertAtEnd(DATA)  
6. CASE 3: GET DATA, GET POSITION, CALL insertAtPosition(DATA, POSITION)  
7. CASE 4: CALL deleteAtBeginning()  
8. CASE 5: CALL deleteAtEnd()  
9. CASE 6: GET POSITION, CALL deleteAtPosition(POSITION)  
10. CASE 7: CALL display()  
11. CASE 8: EXIT  
12. DEFAULT: PRINT "Invalid choice"  
 [END OF SWITCH]  
13. GOTO Step 2  
```

### 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;  
 int i = 1;  
 while (i < position - 1 && temp != NULL) {  
 temp = temp->next;  
 i++;  
 }  
 if (temp == NULL) {  
 printf("Invalid position\n");  
 return;  
 }  
 newNode->next = temp->next;  
 temp->next = newNode;  
}  
  
void deleteAtBeginning() {  
 if (head == NULL) {  
 printf("List is empty\n");  
 return;  
 }  
 struct Node \*temp = head;  
 head = head->next;  
 free(temp);  
}  
  
void deleteAtEnd() {  
 if (head == NULL) {  
 printf("List is empty\n");  
 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 deleteAtPosition(int position) {  
 if (head == NULL) {  
 printf("List is empty\n");  
 return;  
 }  
 if (position == 1) {  
 deleteAtBeginning();  
 return;  
 }  
 struct Node \*temp = head;  
 int i = 1;  
 while (i < position - 1 && temp != NULL) {  
 temp = temp->next;  
 i++;  
 }  
 if (temp == NULL || temp->next == NULL) {  
 printf("Invalid position\n");  
 return;  
 }  
 struct Node \*nodeToDelete = temp->next;  
 temp->next = temp->next->next;  
 free(nodeToDelete);  
}  
  
void display() {  
 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;  
 while (1) {  
 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 at Beginning\n");  
 printf("5. Delete at End\n");  
 printf("6. Delete at Position\n");  
 printf("7. Display\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:  
 deleteAtBeginning();  
 break;  
 case 5:  
 deleteAtEnd();  
 break;  
 case 6:  
 printf("Enter position to delete from: ");  
 scanf("%d", &position);  
 deleteAtPosition(position);  
 break;  
 case 7:  
 display();  
 break;  
 case 8:  
 exit(0);  
 default:  
 printf("Invalid choice\n");  
 }  
 }  
 return 0;  
}

### Output:



## 19. Traverse the list to display each element

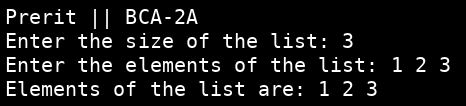
### Algorithm

Here's the algorithm for the provided C code, following the specified formatting guidelines:  
  
1. PRINT "Prerit || BCA-2A"  
2. INPUT size  
3. DECLARE list AS INTEGER ARRAY OF SIZE size  
4. PRINT "Enter the elements of the list: "  
5. SET i = 0  
6. WHILE i < size  
 \* INPUT list\[i]  
 \* SET i = i + 1  
 [END OF LOOP]  
7. PRINT "Elements of the list are: "  
8. SET i = 0  
9. WHILE i < size  
 \* PRINT list\[i] followed by a space  
 \* SET i = i + 1  
 [END OF LOOP]  
10. PRINT a newline character  
11. EXIT

### 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:



## 20. Search for a specific element in the list

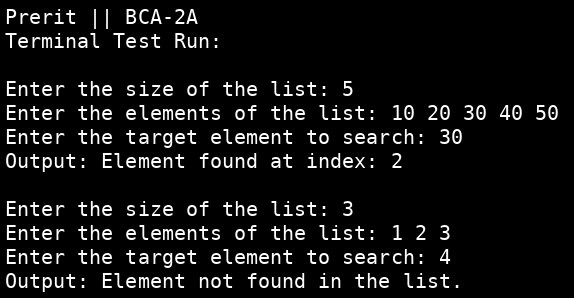
### Algorithm

Here is the algorithm in a clean, textbook style, based on the provided C code.  
  
1. START  
2. DISPLAY "Prerit || BCA-2A"  
3. DECLARE integer variables \*n\*, \*target\*, \*i\*, \*found\*, \*index\*  
4. SET \*found\* = 0  
5. SET \*index\* = -1  
6. DISPLAY "Enter the size of the list: "  
7. READ \*n\*  
8. DECLARE an integer array \*list\* of size \*n\*  
9. DISPLAY "Enter the elements of the list: "  
10. FOR \*i\* = 0 TO \*n\* - 1 DO  
 1. READ \*list\*[i]  
 [END OF LOOP]  
11. DISPLAY "Enter the target element to search: "  
12. READ \*target\*  
13. FOR \*i\* = 0 TO \*n\* - 1 DO  
 1. IF \*list\*[i] == \*target\* THEN  
 1. SET \*found\* = 1  
 2. SET \*index\* = \*i\*  
 3. EXIT FOR loop  
 [END OF IF]  
 [END OF LOOP]  
14. IF \*found\* == 1 THEN  
 1. DISPLAY "Element found at index: ", \*index\*  
 [END OF IF]  
15. ELSE  
 1. DISPLAY "Element not found in the list."  
 [END OF ELSE]  
16. EXIT

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, target, i, found = 0, index = -1;  
  
 printf("Enter the size of the list: ");  
 scanf("%d", &n);  
  
 int list[n];  
  
 printf("Enter the elements of the list: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &list[i]);  
 }  
  
 printf("Enter the target element to search: ");  
 scanf("%d", &target);  
  
 for (i = 0; i < n; i++) {  
 if (list[i] == target) {  
 found = 1;  
 index = i;  
 break;  
 }  
 }  
  
 if (found == 1) {  
 printf("Element found at index: %d\n", index);  
 } else {  
 printf("Element not found in the list.\n");  
 }  
  
 return 0;  
}

### Output:



## 21. WAP to implement Singly Linked List that stores data as integer and perform following operations: (a.) Count the number of nodes in the list (b.) Find minimum and maximum value in the linked list.

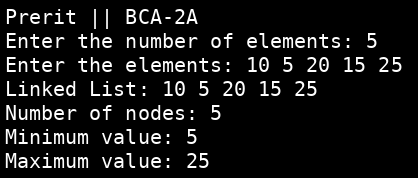
### Algorithm

```  
// Algorithm for Linked List Operations  
  
// InsertNode(data) Algorithm  
1. SET newNode = allocate memory for a Node  
2. SET newNode→DATA = data  
3. SET newNode→NEXT = NULL  
4. IF head == NULL THEN  
 1. SET head = newNode  
 [END OF IF]  
5. ELSE  
 1. SET temp = head  
 2. WHILE temp→NEXT != NULL DO  
 1. SET temp = temp→NEXT  
 [END OF LOOP]  
 3. SET temp→NEXT = newNode  
 [END OF ELSE]  
6. EXIT  
  
// CountNodes() Algorithm  
1. SET count = 0  
2. SET temp = head  
3. WHILE temp != NULL DO  
 1. INCREMENT count  
 2. SET temp = temp→NEXT  
 [END OF LOOP]  
4. RETURN count  
  
// FindMinMax() Algorithm  
1. IF head == NULL THEN  
 1. PRINT "List is empty."  
 2. EXIT  
 [END OF IF]  
2. SET minVal = head→DATA  
3. SET maxVal = head→DATA  
4. SET temp = head→NEXT  
5. WHILE temp != NULL DO  
 1. IF temp→DATA < minVal THEN  
 1. SET minVal = temp→DATA  
 [END OF IF]  
 2. IF temp→DATA > maxVal THEN  
 1. SET maxVal = temp→DATA  
 [END OF IF]  
 3. SET temp = temp→NEXT  
 [END OF LOOP]  
6. PRINT "Minimum value: ", minVal  
7. PRINT "Maximum value: ", maxVal  
8. EXIT  
  
// DisplayList() Algorithm  
1. IF head == NULL THEN  
 1. PRINT "List is empty."  
 2. EXIT  
 [END OF IF]  
2. SET temp = head  
3. PRINT "Linked List: "  
4. WHILE temp != NULL DO  
 1. PRINT temp→DATA, " "  
 2. SET temp = temp→NEXT  
 [END OF LOOP]  
5. PRINT newline  
6. EXIT  
  
// Main() Algorithm  
1. PRINT "Prerit || BCA-2A"  
2. READ n (number of elements)  
3. PRINT "Enter the elements: "  
4. FOR i = 0 TO n-1  
 1. READ data  
 2. CALL InsertNode(data)  
[END OF LOOP]  
5. CALL DisplayList()  
6. SET count = CALL CountNodes()  
7. PRINT "Number of nodes: ", count  
8. CALL FindMinMax()  
9. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
};  
  
struct Node \*head = NULL;  
  
void insertNode(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;  
}  
  
int countNodes() {  
 int count = 0;  
 struct Node \*temp = head;  
 while (temp != NULL) {  
 count++;  
 temp = temp->next;  
 }  
 return count;  
}  
  
void findMinMax() {  
 if (head == NULL) {  
 printf("List is empty.\n");  
 return;  
 }  
  
 int minVal = head->data;  
 int maxVal = head->data;  
  
 struct Node \*temp = head->next;  
 while (temp != NULL) {  
 if (temp->data < minVal) {  
 minVal = temp->data;  
 }  
 if (temp->data > maxVal) {  
 maxVal = temp->data;  
 }  
 temp = temp->next;  
 }  
  
 printf("Minimum value: %d\n", minVal);  
 printf("Maximum value: %d\n", maxVal);  
}  
void displayList() {  
 if (head == NULL) {  
 printf("List is empty.\n");  
 return;  
 }  
  
 struct Node \*temp = head;  
 printf("Linked List: ");  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, data, count;  
 printf("\nEnter the number of elements: ");  
 scanf("%d", &n);  
  
 printf("Enter the elements: ");  
 for (int i = 0; i < n; i++) {  
 scanf("%d", &data);  
 insertNode(data);  
 }  
  
 displayList();  
  
 count = countNodes();  
 printf("Number of nodes: %d\n", count);  
  
 findMinMax();  
  
 return 0;  
}

### Output:



## 22. WAP to implement Singly Linked List that stores data as integer and perform following operations: (a.) Insert a new node in the beginning and end of the list (b.) Insert a new node after a given node in the list. (c.) Insert a new node before a given node in the list.

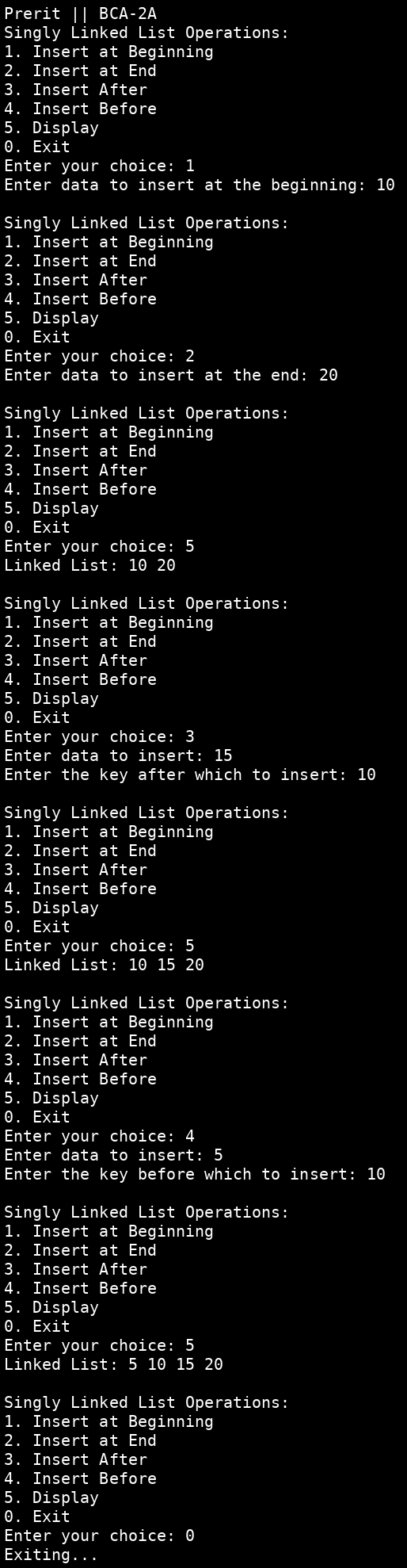
### Algorithm

```  
// Algorithm for Singly Linked List Operations  
  
// Initialize global variable  
1. SET HEAD = NULL  
  
// Algorithm for insertBeginning(DATA)  
2. ALLOCATE NEWNODE  
3. NEWNODE→DATA = DATA  
4. NEWNODE→NEXT = HEAD  
5. HEAD = NEWNODE  
6. EXIT  
  
// Algorithm for insertEnd(DATA)  
7. ALLOCATE NEWNODE  
8. NEWNODE→DATA = DATA  
9. NEWNODE→NEXT = NULL  
10. IF HEAD == NULL THEN  
 11. HEAD = NEWNODE  
 12. EXIT  
 [END OF IF]  
13. SET TEMP = HEAD  
14. WHILE TEMP→NEXT != NULL DO  
 15. TEMP = TEMP→NEXT  
 [END OF LOOP]  
16. TEMP→NEXT = NEWNODE  
17. EXIT  
  
// Algorithm for insertAfter(DATA, KEY)  
18. SET TEMP = HEAD  
19. WHILE TEMP != NULL AND TEMP→DATA != KEY DO  
 20. TEMP = TEMP→NEXT  
 [END OF LOOP]  
21. IF TEMP == NULL THEN  
 22. PRINT "Node not found"  
 23. EXIT  
 [END OF IF]  
24. ALLOCATE NEWNODE  
25. NEWNODE→DATA = DATA  
26. NEWNODE→NEXT = TEMP→NEXT  
27. TEMP→NEXT = NEWNODE  
28. EXIT  
  
// Algorithm for insertBefore(DATA, KEY)  
29. IF HEAD == NULL THEN  
 30. PRINT "List is empty"  
 31. EXIT  
 [END OF IF]  
32. IF HEAD→DATA == KEY THEN  
 33. ALLOCATE NEWNODE  
 34. NEWNODE→DATA = DATA  
 35. NEWNODE→NEXT = HEAD  
 36. HEAD = NEWNODE  
 37. EXIT  
 [END OF IF]  
38. SET TEMP = HEAD  
39. SET PREV = NULL  
40. WHILE TEMP != NULL AND TEMP→DATA != KEY DO  
 41. PREV = TEMP  
 42. TEMP = TEMP→NEXT  
 [END OF LOOP]  
43. IF TEMP == NULL THEN  
 44. PRINT "Node not found"  
 45. EXIT  
 [END OF IF]  
46. ALLOCATE NEWNODE  
47. NEWNODE→DATA = DATA  
48. NEWNODE→NEXT = TEMP  
49. PREV→NEXT = NEWNODE  
50. EXIT  
  
// Algorithm for display()  
51. SET TEMP = HEAD  
52. WHILE TEMP != NULL DO  
 53. PRINT TEMP→DATA  
 54. TEMP = TEMP→NEXT  
 [END OF LOOP]  
55. PRINT newline  
56. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
};  
  
struct Node \*head = NULL;  
  
void insertBeginning(int data) {  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = head;  
 head = newNode;  
}  
  
void insertEnd(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 insertAfter(int data, int key) {  
 struct Node \*temp = head;  
 while (temp != NULL && temp->data != key) {  
 temp = temp->next;  
 }  
  
 if (temp == NULL) {  
 printf("Node not found\n");  
 return;  
 }  
  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = temp->next;  
 temp->next = newNode;  
}  
  
void insertBefore(int data, int key) {  
 if (head == NULL) {  
 printf("List is empty\n");  
 return;  
 }  
 if (head->data == key){  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = head;  
 head = newNode;  
 return;  
 }  
  
 struct Node \*temp = head;  
 struct Node \*prev = NULL;  
  
 while (temp != NULL && temp->data != key) {  
 prev = temp;  
 temp = temp->next;  
 }  
  
 if (temp == NULL) {  
 printf("Node not found\n");  
 return;  
 }  
  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = temp;  
 prev->next = newNode;  
}  
  
void display() {  
 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, key;  
  
 do {  
 printf("\nSingly Linked List Operations:\n");  
 printf("1. Insert at Beginning\n");  
 printf("2. Insert at End\n");  
 printf("3. Insert After\n");  
 printf("4. Insert Before\n");  
 printf("5. Display\n");  
 printf("0. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to insert at the beginning: ");  
 scanf("%d", &data);  
 insertBeginning(data);  
 break;  
 case 2:  
 printf("Enter data to insert at the end: ");  
 scanf("%d", &data);  
 insertEnd(data);  
 break;  
 case 3:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 printf("Enter the key after which to insert: ");  
 scanf("%d", &key);  
 insertAfter(data, key);  
 break;  
 case 4:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 printf("Enter the key before which to insert: ");  
 scanf("%d", &key);  
 insertBefore(data, key);  
 break;  
 case 5:  
 printf("Linked List: ");  
 display();  
 break;  
 case 0:  
 printf("Exiting...\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 0);  
  
 return 0;  
}

### Output:



## 23. WAP to implement two Singly Linked List that stores data as integer and perform following operation: (a.) Merge the two lists to create a new sorted list

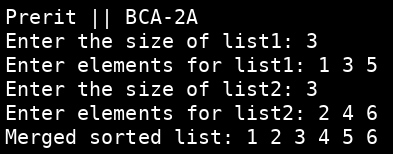
### Algorithm

```  
// Algorithm for merging two sorted linked lists  
  
// Function: MERGE\_SORTED\_LISTS (LIST1, LIST2)  
// Input: Two sorted linked lists LIST1 and LIST2.  
// Output: A new sorted linked list containing all elements from LIST1 and LIST2.  
  
1. SET MERGED\_LIST = NULL  
2. SET TAIL = NULL  
3. WHILE (LIST1 != NULL AND LIST2 != NULL)  
 4. IF (LIST1→DATA <= LIST2→DATA)  
 5. SET NEW\_NODE = LIST1  
 6. SET LIST1 = LIST1→NEXT  
 [END OF IF]  
 7. ELSE  
 8. SET NEW\_NODE = LIST2  
 9. SET LIST2 = LIST2→NEXT  
 [END OF ELSE]  
 10. IF (MERGED\_LIST == NULL)  
 11. SET MERGED\_LIST = NEW\_NODE  
 12. SET TAIL = NEW\_NODE  
 [END OF IF]  
 13. ELSE  
 14. SET TAIL→NEXT = NEW\_NODE  
 15. SET TAIL = NEW\_NODE  
 [END OF ELSE]  
 16. SET NEW\_NODE→NEXT = NULL  
 [END OF LOOP]  
4. IF (LIST1 != NULL)  
 18. IF (MERGED\_LIST == NULL)  
 19. SET MERGED\_LIST = LIST1  
 [END OF IF]  
 20. ELSE  
 21. SET TAIL→NEXT = LIST1  
 [END OF ELSE]  
 [END OF IF]  
5. IF (LIST2 != NULL)  
 23. IF (MERGED\_LIST == NULL)  
 24. SET MERGED\_LIST = LIST2  
 [END OF IF]  
 25. ELSE  
 26. SET TAIL→NEXT = LIST2  
 [END OF ELSE]  
 [END OF IF]  
6. RETURN MERGED\_LIST  
7. EXIT  
  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
};  
  
struct Node \*insertAtEnd(struct Node \*list, int data) {  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = NULL;  
 if (list == NULL) {  
 return newNode;  
 }  
 struct Node \*temp = list;  
 while (temp->next != NULL) {  
 temp = temp->next;  
 }  
 temp->next = newNode;  
 return list;  
}  
  
struct Node \*createList(struct Node \*list, int size) {  
 int data;  
 for (int i = 0; i < size; i++) {  
 scanf("%d", &data);  
 list = insertAtEnd(list, data);  
 }  
 return list;  
}  
  
struct Node \*mergeSortedLists(struct Node \*list1, struct Node \*list2) {  
 struct Node \*mergedList = NULL;  
 struct Node \*tail = NULL;  
  
 while (list1 != NULL && list2 != NULL) {  
 struct Node \*newNode;  
 if (list1->data <= list2->data) {  
 newNode = list1;  
 list1 = list1->next;  
 } else {  
 newNode = list2;  
 list2 = list2->next;  
 }  
  
 if (mergedList == NULL) {  
 mergedList = newNode;  
 tail = newNode;  
 } else {  
 tail->next = newNode;  
 tail = newNode;  
 }  
 newNode->next = NULL;  
 }  
  
 if (list1 != NULL) {  
 if (mergedList == NULL) {  
 mergedList = list1;  
 } else {  
 tail->next = list1;  
 }  
 }  
  
 if (list2 != NULL) {  
 if (mergedList == NULL) {  
 mergedList = list2;  
 } else {  
 tail->next = list2;  
 }  
 }  
  
 return mergedList;  
}  
  
  
void printList(struct Node \*list) {  
 struct Node \*temp = list;  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
void freeList(struct Node \*list) {  
 struct Node \*temp;  
 while (list != NULL) {  
 temp = list;  
 list = list->next;  
 free(temp);  
 }  
}  
  
  
int main() {  
 printf("Prerit || BCA-2A");  
 struct Node \*list1 = NULL;  
 struct Node \*list2 = NULL;  
 struct Node \*mergedList = NULL;  
 int size1, size2;  
  
 printf("Enter the size of list1: ");  
 scanf("%d", &size1);  
 printf("Enter elements for list1: ");  
 list1 = createList(list1, size1);  
  
 printf("Enter the size of list2: ");  
 scanf("%d", &size2);  
 printf("Enter elements for list2: ");  
 list2 = createList(list2, size2);  
  
 mergedList = mergeSortedLists(list1, list2);  
  
 printf("Merged sorted list: ");  
 printList(mergedList);  
  
 freeList(list1);  
 freeList(list2);  
 freeList(mergedList);  
  
 return 0;  
}

### Output:



## 24. WAP to implement a Singly Linked List that stores data as integer and perform following operation: (a.) Create a new list that is reverse of the first linked list

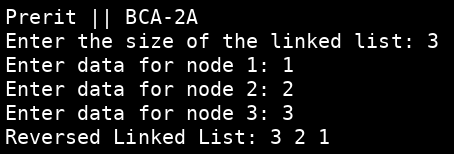
### Algorithm

```  
ALGORITHM ReverseLinkedList  
  
// Main Function  
1. INPUT n (size of the linked list)  
2. SET head = CREATE\_LIST(n)  
3. SET reversedHead = REVERSE\_LIST(head)  
4. OUTPUT "Reversed Linked List: "  
5. CALL DISPLAY\_LIST(reversedHead)  
6. EXIT  
  
// Function: CREATE\_LIST(n)  
1. SET head = NULL, temp = NULL, newNode = NULL  
2. ALLOCATE memory for head  
3. OUTPUT "Enter data for node 1: "  
4. INPUT head→DATA  
5. SET head→NEXT = NULL  
6. SET temp = head  
7. FOR i = 2 TO n  
 8. ALLOCATE memory for newNode  
 9. OUTPUT "Enter data for node ", i, ": "  
 10. INPUT newNode→DATA  
 11. SET newNode→NEXT = NULL  
 12. SET temp→NEXT = newNode  
 13. SET temp = newNode  
 [END OF LOOP]  
8. RETURN head  
  
// Function: REVERSE\_LIST(head)  
1. SET prev = NULL, current = head, next = NULL  
2. WHILE current != NULL  
 3. SET next = current→NEXT  
 4. SET current→NEXT = prev  
 5. SET prev = current  
 6. SET current = next  
 [END OF LOOP]  
3. RETURN prev  
  
// Function: DISPLAY\_LIST(head)  
1. SET temp = head  
2. WHILE temp != NULL  
 3. OUTPUT temp→DATA, " "  
 4. SET temp = temp→NEXT  
 [END OF LOOP]  
3. OUTPUT newline  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
};  
  
struct Node\* createList(int n) {  
 struct Node \*head = NULL, \*temp = NULL, \*newNode = NULL;  
 int i;  
  
 head = (struct Node\*)malloc(sizeof(struct Node));  
 printf("Enter data for node 1: ");  
 scanf("%d", &(head->data));  
 head->next = NULL;  
 temp = head;  
  
 for (i = 2; i <= n; i++) {  
 newNode = (struct Node\*)malloc(sizeof(struct Node));  
 printf("Enter data for node %d: ", i);  
 scanf("%d", &(newNode->data));  
 newNode->next = NULL;  
 temp->next = newNode;  
 temp = newNode;  
 }  
 return head;  
}  
  
struct Node\* reverseList(struct Node \*head) {  
 struct Node \*prev = NULL, \*current = head, \*next = NULL;  
 while (current != NULL) {  
 next = current->next;  
 current->next = prev;  
 prev = current;  
 current = next;  
 }  
 return prev;  
}  
  
void displayList(struct Node \*head) {  
 struct Node \*temp = head;  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
int main() {  
 Prerit || BCA-2A  
 int n;  
 struct Node \*head = NULL, \*reversedHead = NULL;  
  
 printf("Enter the size of the linked list: ");  
 scanf("%d", &n);  
  
 head = createList(n);  
 reversedHead = reverseList(head);  
  
 printf("Reversed Linked List: ");  
 displayList(reversedHead);  
  
 return 0;  
}

### Output:



## 25. WAP to implement Singly Linked List that stores data as integer and perform following operations: (a.) Delete a node in the beginning and end of the list. (b.) Delete the node that comes after a given node in the linked list. (c.) Search an element in the linked list. If found, delete it.

### Algorithm

```  
// Algorithm for Singly Linked List Operations  
  
// Data Structures:  
// NODE: Structure containing data and pointer to the next node.  
// HEAD: Pointer to the first node of the list (global).  
  
// Function: insertAtEnd(data)  
// Inserts a new node with the given data at the end of the list.  
1. ALLOCATE newNode: NODE  
2. SET newNode→DATA = data  
3. SET newNode→NEXT = NULL  
4. IF HEAD is NULL THEN  
 5. SET HEAD = newNode  
 6. EXIT  
 [END OF IF]  
7. SET temp = HEAD  
8. WHILE temp→NEXT is not NULL DO  
 9. SET temp = temp→NEXT  
 [END OF LOOP]  
10. SET temp→NEXT = newNode  
11. EXIT  
  
// Function: displayList()  
// Displays the data of each node in the list.  
1. SET temp = HEAD  
2. WHILE temp is not NULL DO  
 3. PRINT temp→DATA  
 4. SET temp = temp→NEXT  
 [END OF LOOP]  
5. PRINT newline character  
6. EXIT  
  
// Function: deleteAtBeginning()  
// Deletes the node at the beginning of the list.  
1. IF HEAD is NULL THEN  
 2. EXIT  
 [END OF IF]  
3. SET temp = HEAD  
4. SET HEAD = HEAD→NEXT  
5. FREE temp  
6. EXIT  
  
// Function: deleteAtEnd()  
// Deletes the node at the end of the list.  
1. IF HEAD is NULL THEN  
 2. EXIT  
 [END OF IF]  
3. IF HEAD→NEXT is NULL THEN  
 4. FREE HEAD  
 5. SET HEAD = NULL  
 6. EXIT  
 [END OF IF]  
7. SET current = HEAD  
8. SET previous = NULL  
9. WHILE current→NEXT is not NULL DO  
 10. SET previous = current  
 11. SET current = current→NEXT  
 [END OF LOOP]  
12. SET previous→NEXT = NULL  
13. FREE current  
14. EXIT  
  
// Function: deleteAfterGivenNode(givenData)  
// Deletes the node immediately after the node containing givenData.  
1. IF HEAD is NULL THEN  
 2. EXIT  
 [END OF IF]  
3. SET temp = HEAD  
4. WHILE temp is not NULL AND temp→DATA is not equal to givenData DO  
 5. SET temp = temp→NEXT  
 [END OF LOOP]  
6. IF temp is NULL THEN  
 7. PRINT "Node not found."  
 8. EXIT  
 [END OF IF]  
9. IF temp→NEXT is NULL THEN  
 10. PRINT "No node after the given node."  
 11. EXIT  
 [END OF IF]  
12. SET nextNode = temp→NEXT  
13. SET temp→NEXT = nextNode→NEXT  
14. FREE nextNode  
15. EXIT  
  
// Function: searchAndDelete(key)  
// Searches for the node containing key and deletes it.  
1. SET temp = HEAD  
2. SET prev = NULL  
  
3. IF temp is not NULL AND temp→DATA is equal to key THEN  
 4. SET HEAD = temp→NEXT  
 5. FREE temp  
 6. EXIT  
 [END OF IF]  
  
7. WHILE temp is not NULL AND temp→DATA is not equal to key DO  
 8. SET prev = temp  
 9. SET temp = temp→NEXT  
 [END OF LOOP]  
  
10. IF temp is NULL THEN  
 11. PRINT "Element not found."  
 12. EXIT  
 [END OF IF]  
  
13. SET prev→NEXT = temp→NEXT  
14. FREE temp  
15. EXIT  
  
//Main Function is a interactive menu for the user.  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
};  
  
struct Node \*head = NULL;  
  
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 displayList() {  
 struct Node \*temp = head;  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
void deleteAtBeginning() {  
 if (head == NULL) return;  
 struct Node \*temp = head;  
 head = head->next;  
 free(temp);  
}  
  
void deleteAtEnd() {  
 if (head == NULL) return;  
 if (head->next == NULL) {  
 free(head);  
 head = NULL;  
 return;  
 }  
 struct Node \*current = head;  
 struct Node \*previous = NULL;  
 while (current->next != NULL) {  
 previous = current;  
 current = current->next;  
 }  
 previous->next = NULL;  
 free(current);  
}  
  
void deleteAfterGivenNode(int givenData) {  
 if (head == NULL) return;  
 struct Node \*temp = head;  
 while (temp != NULL && temp->data != givenData) {  
 temp = temp->next;  
 }  
 if (temp == NULL) {  
 printf("Node not found.\n");  
 return;  
 }  
 if (temp->next == NULL) {  
 printf("No node after the given node.\n");  
 return;  
 }  
 struct Node \*nextNode = temp->next;  
 temp->next = nextNode->next;  
 free(nextNode);  
}  
  
void searchAndDelete(int key) {  
 struct Node \*temp = head;  
 struct Node \*prev = NULL;  
  
 if (temp != NULL && temp->data == key) {  
 head = temp->next;  
 free(temp);  
 return;  
 }  
  
 while (temp != NULL && temp->data != key) {  
 prev = temp;  
 temp = temp->next;  
 }  
  
 if (temp == NULL) {  
 printf("Element not found.\n");  
 return;  
 }  
  
 prev->next = temp->next;  
 free(temp);  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, data, givenData, key;  
 while (1) {  
 printf("\nSingly Linked List Operations:\n");  
 printf("1. Insert at End\n");  
 printf("2. Display List\n");  
 printf("3. Delete at Beginning\n");  
 printf("4. Delete at End\n");  
 printf("5. Delete After Given Node\n");  
 printf("6. Search and Delete\n");  
 printf("7. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
 switch (choice) {  
 case 1:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 insertAtEnd(data);  
 break;  
 case 2:  
 printf("Linked List: ");  
 displayList();  
 break;  
 case 3:  
 deleteAtBeginning();  
 printf("Deleted node at beginning.\n");  
 break;  
 case 4:  
 deleteAtEnd();  
 printf("Deleted node at end.\n");  
 break;  
 case 5:  
 printf("Enter data of the node after which to delete: ");  
 scanf("%d", &givenData);  
 deleteAfterGivenNode(givenData);  
 break;  
 case 6:  
 printf("Enter element to search and delete: ");  
 scanf("%d", &key);  
 searchAndDelete(key);  
 break;  
 case 7:  
 exit(0);  
 default:  
 printf("Invalid choice.\n");  
 }  
 }  
 return 0;  
}

### Output:



## 26. WAP to implement Doubly Linked List that stores data as integer and perform following operations: (a.) Traverse the list to display each element (b.) Search for a specific element in the list

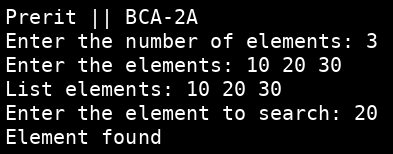
### Algorithm

```  
// Algorithm for Doubly Linked List Operations  
  
// Initialization  
1. SET HEAD = NULL  
  
// Function: insertAtEnd(DATA)  
2. CREATE NEWNODE  
3. NEWNODE→DATA = DATA  
4. NEWNODE→NEXT = NULL  
5. NEWNODE→PREV = NULL  
6. IF HEAD == NULL THEN  
 HEAD = NEWNODE  
 EXIT FUNCTION  
 [END OF IF]  
7. SET TEMP = HEAD  
8. WHILE TEMP→NEXT != NULL DO  
 TEMP = TEMP→NEXT  
 [END OF LOOP]  
9. TEMP→NEXT = NEWNODE  
10. NEWNODE→PREV = TEMP  
11. EXIT FUNCTION  
  
// Function: traverseList()  
12. IF HEAD == NULL THEN  
 PRINT "List is empty"  
 EXIT FUNCTION  
 [END OF IF]  
13. SET TEMP = HEAD  
14. PRINT "List elements: "  
15. WHILE TEMP != NULL DO  
 PRINT TEMP→DATA  
 TEMP = TEMP→NEXT  
 [END OF LOOP]  
16. PRINT NEWLINE  
17. EXIT FUNCTION  
  
// Function: searchElement(KEY)  
18. IF HEAD == NULL THEN  
 PRINT "List is empty"  
 EXIT FUNCTION  
 [END OF IF]  
19. SET TEMP = HEAD  
20. WHILE TEMP != NULL DO  
 IF TEMP→DATA == KEY THEN  
 PRINT "Element found"  
 EXIT FUNCTION  
 [END OF IF]  
 TEMP = TEMP→NEXT  
 [END OF LOOP]  
21. PRINT "Element not found"  
22. EXIT FUNCTION  
  
// Main Function  
23. PRINT "Prerit || BCA-2A"  
24. INPUT N  
25. PRINT "Enter the elements: "  
26. FOR I = 0 TO N-1 DO  
 INPUT DATA  
 CALL insertAtEnd(DATA)  
 [END OF LOOP]  
27. CALL traverseList()  
28. PRINT "Enter the element to search: "  
29. INPUT KEY  
30. CALL searchElement(KEY)  
31. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
 struct Node \*prev;  
};  
  
struct Node \*head = NULL;  
  
void insertAtEnd(int data) {  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->next = NULL;  
 newNode->prev = NULL;  
  
 if (head == NULL) {  
 head = newNode;  
 return;  
 }  
  
 struct Node \*temp = head;  
 while (temp->next != NULL) {  
 temp = temp->next;  
 }  
  
 temp->next = newNode;  
 newNode->prev = temp;  
}  
  
void traverseList() {  
 if (head == NULL) {  
 printf("List is empty\n");  
 return;  
 }  
  
 struct Node \*temp = head;  
 printf("List elements: ");  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
void searchElement(int key) {  
 if (head == NULL) {  
 printf("List is empty\n");  
 return;  
 }  
  
 struct Node \*temp = head;  
 while (temp != NULL) {  
 if (temp->data == key) {  
 printf("Element found\n");  
 return;  
 }  
 temp = temp->next;  
 }  
  
 printf("Element not found\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, data, key;  
  
 printf("\nEnter the number of elements: ");  
 scanf("%d", &n);  
  
 printf("Enter the elements: ");  
 for (int i = 0; i < n; i++) {  
 scanf("%d", &data);  
 insertAtEnd(data);  
 }  
  
 traverseList();  
  
 printf("Enter the element to search: ");  
 scanf("%d", &key);  
  
 searchElement(key);  
  
 return 0;  
}

### Output:



## 27. WAP to implement Doubly Linked List that stores data as integer and perform following operations: (a.) Insert a new node in the beginning, end and middle of the list (b.) Delete a node in the beginning, end and middle of the list (c.) WAP to implement Header Linked List with operations: (d.) Insertion(Beginning, Between, End) (e.) Deletion(Beginning, Between, End) (f.) Traverse

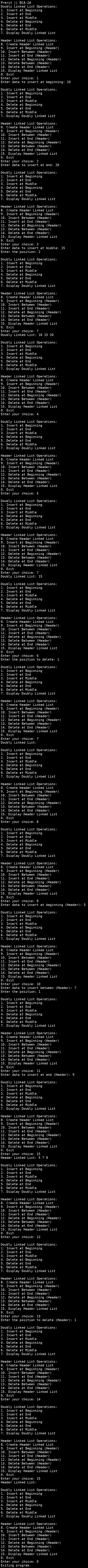
### Algorithm

```text  
// Algorithm for Doubly Linked List and Header Linked List Operations  
  
// Doubly Linked List Operations  
  
// Algorithm: insertDoublyBeginning(data)  
1. CREATE newNode.  
2. SET newNode→DATA = data.  
3. SET newNode→PREV = NULL.  
4. SET newNode→NEXT = doublyHead.  
5. IF doublyHead == NULL THEN  
 \* SET doublyHead = newNode.  
 \* SET doublyTail = newNode.  
6. ELSE  
 \* SET doublyHead→PREV = newNode.  
 \* SET doublyHead = newNode.  
7. [END OF IF]  
8. EXIT  
  
// Algorithm: insertDoublyEnd(data)  
1. CREATE newNode.  
2. SET newNode→DATA = data.  
3. SET newNode→NEXT = NULL.  
4. SET newNode→PREV = doublyTail.  
5. IF doublyTail == NULL THEN  
 \* SET doublyHead = newNode.  
 \* SET doublyTail = newNode.  
6. ELSE  
 \* SET doublyTail→NEXT = newNode.  
 \* SET doublyTail = newNode.  
7. [END OF IF]  
8. EXIT  
  
// Algorithm: insertDoublyMiddle(data, position)  
1. IF doublyHead == NULL THEN  
 \* PRINT "List is empty."  
 \* EXIT  
2. [END OF IF]  
3. CREATE newNode.  
4. SET newNode→DATA = data.  
5. IF position == 1 THEN  
 \* SET newNode→NEXT = doublyHead.  
 \* SET newNode→PREV = NULL.  
 \* SET doublyHead→PREV = newNode.  
 \* SET doublyHead = newNode.  
 \* EXIT  
6. [END OF IF]  
7. SET current = doublyHead.  
8. SET count = 1.  
9. WHILE current != NULL AND count < position - 1 DO  
 \* SET current = current→NEXT.  
 \* SET count = count + 1.  
10. [END OF LOOP]  
11. IF current == NULL THEN  
 \* PRINT "Position out of bounds."  
 \* FREE newNode  
 \* EXIT  
12. [END OF IF]  
13. SET newNode→NEXT = current→NEXT.  
14. SET newNode→PREV = current.  
15. IF current→NEXT != NULL THEN  
 \* SET current→NEXT→PREV = newNode.  
16. ELSE  
 \* SET doublyTail = newNode.  
17. [END OF IF]  
18. SET current→NEXT = newNode.  
19. EXIT  
  
// Algorithm: deleteDoublyBeginning()  
1. IF doublyHead == NULL THEN  
 \* PRINT "List is empty."  
 \* EXIT  
2. [END OF IF]  
3. SET temp = doublyHead.  
4. SET doublyHead = doublyHead→NEXT.  
5. IF doublyHead != NULL THEN  
 \* SET doublyHead→PREV = NULL.  
6. ELSE  
 \* SET doublyTail = NULL.  
7. [END OF IF]  
8. FREE temp.  
9. EXIT  
  
// Algorithm: deleteDoublyEnd()  
1. IF doublyTail == NULL THEN  
 \* PRINT "List is empty."  
 \* EXIT  
2. [END OF IF]  
3. SET temp = doublyTail.  
4. SET doublyTail = doublyTail→PREV.  
5. IF doublyTail != NULL THEN  
 \* SET doublyTail→NEXT = NULL.  
6. ELSE  
 \* SET doublyHead = NULL.  
7. [END OF IF]  
8. FREE temp.  
9. EXIT  
  
// Algorithm: deleteDoublyMiddle(position)  
1. IF doublyHead == NULL THEN  
 \* PRINT "List is empty."  
 \* EXIT  
2. [END OF IF]  
3. SET current = doublyHead.  
4. SET count = 1.  
5. IF position == 1 THEN  
 \* SET doublyHead = doublyHead→NEXT.  
 \* IF doublyHead != NULL THEN  
 \* SET doublyHead→PREV = NULL.  
 \* ELSE  
 \* SET doublyTail = NULL.  
 \* [END OF IF]  
 \* FREE current.  
 \* EXIT  
6. [END OF IF]  
7. WHILE current != NULL AND count < position DO  
 \* SET current = current→NEXT.  
 \* SET count = count + 1.  
8. [END OF LOOP]  
9. IF current == NULL THEN  
 \* PRINT "Position out of bounds."  
 \* EXIT  
10. [END OF IF]  
11. SET current→PREV→NEXT = current→NEXT.  
12. IF current→NEXT != NULL THEN  
 \* SET current→NEXT→PREV = current→PREV.  
13. ELSE  
 \* SET doublyTail = current→PREV.  
14. [END OF IF]  
15. FREE current.  
16. EXIT  
  
// Algorithm: displayDoubly()  
1. SET current = doublyHead.  
2. PRINT "Doubly Linked List: ".  
3. WHILE current != NULL DO  
 \* PRINT current→DATA.  
 \* SET current = current→NEXT.  
4. [END OF LOOP]  
5. PRINT newline.  
6. EXIT  
  
// Header Linked List Operations  
  
// Algorithm: createHeaderList()  
1. CREATE header.  
2. SET header→DATA = -1.  
3. SET header→NEXT = NULL.  
4. EXIT  
  
// Algorithm: insertHeaderBeginning(data)  
1. CREATE newNode.  
2. SET newNode→DATA = data.  
3. SET newNode→NEXT = header→NEXT.  
4. SET header→NEXT = newNode.  
5. EXIT  
  
// Algorithm: insertHeaderBetween(data, position)  
1. SET current = header.  
2. SET count = 1.  
3. WHILE current != NULL AND count < position DO  
 \* SET current = current→NEXT.  
 \* SET count = count + 1.  
4. [END OF LOOP]  
5. IF current == NULL THEN  
 \* PRINT "Position out of bounds."  
 \* EXIT  
6. [END OF IF]  
7. CREATE newNode.  
8. SET newNode→DATA = data.  
9. SET newNode→NEXT = current→NEXT.  
10. SET current→NEXT = newNode.  
11. EXIT  
  
// Algorithm: insertHeaderEnd(data)  
1. CREATE newNode.  
2. SET newNode→DATA = data.  
3. SET newNode→NEXT = NULL.  
4. SET current = header.  
5. WHILE current→NEXT != NULL DO  
 \* SET current = current→NEXT.  
6. [END OF LOOP]  
7. SET current→NEXT = newNode.  
8. EXIT  
  
// Algorithm: deleteHeaderBeginning()  
1. IF header→NEXT == NULL THEN  
 \* PRINT "List is empty."  
 \* EXIT  
2. [END OF IF]  
3. SET temp = header→NEXT.  
4. SET header→NEXT = temp→NEXT.  
5. FREE temp.  
6. EXIT  
  
// Algorithm: deleteHeaderBetween(position)  
1. SET current = header.  
2. SET count = 1.  
3. WHILE current != NULL AND count < position DO  
 \* SET current = current→NEXT.  
 \* SET count = count + 1.  
4. [END OF LOOP]  
5. IF current == NULL OR current→NEXT == NULL THEN  
 \* PRINT "Position out of bounds."  
 \* EXIT  
6. [END OF IF]  
7. SET temp = current→NEXT.  
8. SET current→NEXT = temp→NEXT.  
9. FREE temp.  
10. EXIT  
  
// Algorithm: deleteHeaderEnd()  
1. IF header→NEXT == NULL THEN  
 \* PRINT "List is empty."  
 \* EXIT  
2. [END OF IF]  
3. SET current = header.  
4. SET prev = NULL.  
5. WHILE current→NEXT != NULL DO  
 \* SET prev = current.  
 \* SET current = current→NEXT.  
6. [END OF LOOP]  
7. SET prev→NEXT = NULL.  
8. FREE current.  
9. EXIT  
  
// Algorithm: displayHeader()  
1. SET current = header→NEXT.  
2. PRINT "Header Linked List: ".  
3. WHILE current != NULL DO  
 \* PRINT current→DATA.  
 \* SET current = current→NEXT.  
4. [END OF LOOP]  
5. PRINT newline.  
6. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct DoublyNode {  
 int data;  
 struct DoublyNode\* next;  
 struct DoublyNode\* prev;  
};  
  
struct HeaderNode {  
 int data;  
 struct HeaderNode\* next;  
};  
  
struct DoublyNode\* doublyHead = NULL;  
struct DoublyNode\* doublyTail = NULL;  
struct HeaderNode\* header = NULL;  
  
void insertDoublyBeginning(int data) {  
 struct DoublyNode\* newNode = (struct DoublyNode\*)malloc(sizeof(struct DoublyNode));  
 newNode->data = data;  
 newNode->prev = NULL;  
 newNode->next = doublyHead;  
  
 if (doublyHead == NULL) {  
 doublyHead = newNode;  
 doublyTail = newNode;  
 } else {  
 doublyHead->prev = newNode;  
 doublyHead = newNode;  
 }  
}  
  
void insertDoublyEnd(int data) {  
 struct DoublyNode\* newNode = (struct DoublyNode\*)malloc(sizeof(struct DoublyNode));  
 newNode->data = data;  
 newNode->next = NULL;  
 newNode->prev = doublyTail;  
  
 if (doublyTail == NULL) {  
 doublyHead = newNode;  
 doublyTail = newNode;  
 } else {  
 doublyTail->next = newNode;  
 doublyTail = newNode;  
 }  
}  
  
void insertDoublyMiddle(int data, int position) {  
 if (doublyHead == NULL) {  
 printf("List is empty. Cannot insert at middle.\n");  
 return;  
 }  
  
 struct DoublyNode\* newNode = (struct DoublyNode\*)malloc(sizeof(struct DoublyNode));  
 newNode->data = data;  
  
 if (position == 1) {  
 newNode->next = doublyHead;  
 newNode->prev = NULL;  
 doublyHead->prev = newNode;  
 doublyHead = newNode;  
 return;  
 }  
  
 struct DoublyNode\* current = doublyHead;  
 int count = 1;  
 while (current != NULL && count < position - 1) {  
 current = current->next;  
 count++;  
 }  
  
 if (current == NULL) {  
 printf("Position out of bounds. Cannot insert at middle.\n");  
 free(newNode);  
 return;  
 }  
  
 newNode->next = current->next;  
 newNode->prev = current;  
 if (current->next != NULL) {  
 current->next->prev = newNode;  
 } else {  
 doublyTail = newNode;  
 }  
 current->next = newNode;  
}  
  
void deleteDoublyBeginning() {  
 if (doublyHead == NULL) {  
 printf("List is empty. Cannot delete from beginning.\n");  
 return;  
 }  
  
 struct DoublyNode\* temp = doublyHead;  
 doublyHead = doublyHead->next;  
  
 if (doublyHead != NULL) {  
 doublyHead->prev = NULL;  
 } else {  
 doublyTail = NULL;  
 }  
  
 free(temp);  
}  
  
void deleteDoublyEnd() {  
 if (doublyTail == NULL) {  
 printf("List is empty. Cannot delete from end.\n");  
 return;  
 }  
  
 struct DoublyNode\* temp = doublyTail;  
 doublyTail = doublyTail->prev;  
  
 if (doublyTail != NULL) {  
 doublyTail->next = NULL;  
 } else {  
 doublyHead = NULL;  
 }  
  
 free(temp);  
}  
  
void deleteDoublyMiddle(int position) {  
 if (doublyHead == NULL) {  
 printf("List is empty. Cannot delete from middle.\n");  
 return;  
 }  
  
 struct DoublyNode\* current = doublyHead;  
 int count = 1;  
  
 if (position == 1) {  
 doublyHead = doublyHead->next;  
 if (doublyHead != NULL) {  
 doublyHead->prev = NULL;  
 } else {  
 doublyTail = NULL;  
 }  
 free(current);  
 return;  
 }  
  
 while (current != NULL && count < position) {  
 current = current->next;  
 count++;  
 }  
  
 if (current == NULL) {  
 printf("Position out of bounds. Cannot delete from middle.\n");  
 return;  
 }  
  
 current->prev->next = current->next;  
 if (current->next != NULL) {  
 current->next->prev = current->prev;  
 } else {  
 doublyTail = current->prev;  
 }  
  
 free(current);  
}  
  
void displayDoubly() {  
 struct DoublyNode\* current = doublyHead;  
 printf("Doubly Linked List: ");  
 while (current != NULL) {  
 printf("%d ", current->data);  
 current = current->next;  
 }  
 printf("\n");  
}  
  
void createHeaderList() {  
 header = (struct HeaderNode\*)malloc(sizeof(struct HeaderNode));  
 header->data = -1;  
 header->next = NULL;  
}  
  
void insertHeaderBeginning(int data) {  
 struct HeaderNode\* newNode = (struct HeaderNode\*)malloc(sizeof(struct HeaderNode));  
 newNode->data = data;  
 newNode->next = header->next;  
 header->next = newNode;  
}  
  
void insertHeaderBetween(int data, int position) {  
 struct HeaderNode\* current = header;  
 int count = 1;  
  
 while (current != NULL && count < position) {  
 current = current->next;  
 count++;  
 }  
  
 if (current == NULL) {  
 printf("Position out of bounds. Cannot insert.\n");  
 return;  
 }  
  
 struct HeaderNode\* newNode = (struct HeaderNode\*)malloc(sizeof(struct HeaderNode));  
 newNode->data = data;  
 newNode->next = current->next;  
 current->next = newNode;  
}  
  
void insertHeaderEnd(int data) {  
 struct HeaderNode\* newNode = (struct HeaderNode\*)malloc(sizeof(struct HeaderNode));  
 newNode->data = data;  
 newNode->next = NULL;  
  
 struct HeaderNode\* current = header;  
 while (current->next != NULL) {  
 current = current->next;  
 }  
 current->next = newNode;  
}  
  
void deleteHeaderBeginning() {  
 if (header->next == NULL) {  
 printf("List is empty. Cannot delete from beginning.\n");  
 return;  
 }  
  
 struct HeaderNode\* temp = header->next;  
 header->next = temp->next;  
 free(temp);  
}  
  
void deleteHeaderBetween(int position) {  
 struct HeaderNode\* current = header;  
 int count = 1;  
  
 while (current != NULL && count < position) {  
 current = current->next;  
 count++;  
 }  
  
 if (current == NULL || current->next == NULL) {  
 printf("Position out of bounds. Cannot delete.\n");  
 return;  
 }  
  
 struct HeaderNode\* temp = current->next;  
 current->next = temp->next;  
 free(temp);  
}  
  
void deleteHeaderEnd() {  
 if (header->next == NULL) {  
 printf("List is empty. Cannot delete from end.\n");  
 return;  
 }  
  
 struct HeaderNode\* current = header;  
 struct HeaderNode\* prev = NULL;  
  
 while (current->next != NULL) {  
 prev = current;  
 current = current->next;  
 }  
  
 prev->next = NULL;  
 free(current);  
}  
  
void displayHeader() {  
 struct HeaderNode\* current = header->next;  
 printf("Header Linked List: ");  
 while (current != NULL) {  
 printf("%d ", current->data);  
 current = current->next;  
 }  
 printf("\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, data, position;  
  
 do {  
 printf("\nDoubly Linked List Operations:\n");  
 printf("1. Insert at Beginning\n");  
 printf("2. Insert at End\n");  
 printf("3. Insert at Middle\n");  
 printf("4. Delete at Beginning\n");  
 printf("5. Delete at End\n");  
 printf("6. Delete at Middle\n");  
 printf("7. Display Doubly Linked List\n");  
 printf("\nHeader Linked List Operations:\n");  
 printf("8. Create Header Linked List\n");  
 printf("9. Insert at Beginning (Header)\n");  
 printf("10. Insert Between (Header)\n");  
 printf("11. Insert at End (Header)\n");  
 printf("12. Delete at Beginning (Header)\n");  
 printf("13. Delete Between (Header)\n");  
 printf("14. Delete at End (Header)\n");  
 printf("15. Display Header Linked List\n");  
 printf("0. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to insert at beginning: ");  
 scanf("%d", &data);  
 insertDoublyBeginning(data);  
 break;  
 case 2:  
 printf("Enter data to insert at end: ");  
 scanf("%d", &data);  
 insertDoublyEnd(data);  
 break;  
 case 3:  
 printf("Enter data to insert at middle: ");  
 scanf("%d", &data);  
 printf("Enter the position: ");  
 scanf("%d", &position);  
 insertDoublyMiddle(data, position);  
 break;  
 case 4:  
 deleteDoublyBeginning();  
 break;  
 case 5:  
 deleteDoublyEnd();  
 break;  
 case 6:  
 printf("Enter the position to delete: ");  
 scanf("%d", &position);  
 deleteDoublyMiddle(position);  
 break;  
 case 7:  
 displayDoubly();  
 break;  
 case 8:  
 createHeaderList();  
 break;  
 case 9:  
 if (header == NULL) {  
 printf("Create Header List first!\n");  
 break;  
 }  
 printf("Enter data to insert at beginning (Header): ");  
 scanf("%d", &data);  
 insertHeaderBeginning(data);  
 break;  
 case 10:  
 if (header == NULL) {  
 printf("Create Header List first!\n");  
 break;  
 }  
 printf("Enter data to insert between (Header): ");  
 scanf("%d", &data);  
 printf("Enter the position: ");  
 scanf("%d", &position);  
 insertHeaderBetween(data, position);  
 break;  
 case 11:  
 if (header == NULL) {  
 printf("Create Header List first!\n");  
 break;  
 }  
 printf("Enter data to insert at end (Header): ");  
 scanf("%d", &data);  
 insertHeaderEnd(data);  
 break;  
 case 12:  
 if (header == NULL) {  
 printf("Create Header List first!\n");  
 break;  
 }  
 deleteHeaderBeginning();  
 break;  
 case 13:  
 if (header == NULL) {  
 printf("Create Header List first!\n");  
 break;  
 }  
 printf("Enter the position to delete (Header): ");  
 scanf("%d", &position);  
 deleteHeaderBetween(position);  
 break;  
 case 14:  
 if (header == NULL) {  
 printf("Create Header List first!\n");  
 break;  
 }  
 deleteHeaderEnd();  
 break;  
 case 15:  
 if (header == NULL) {  
 printf("Create Header List first!\n");  
 break;  
 }  
 displayHeader();  
 break;  
 case 0:  
 printf("Exiting program.\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 0);  
  
 return 0;  
}

### Output:



## 28. WAP to implement Circular Linked List with operations: (a.) Insertion(Beginning, Between, End) (b.) Deletion(Beginning, Between, End) (c.) Traverse (d.) Linear Search

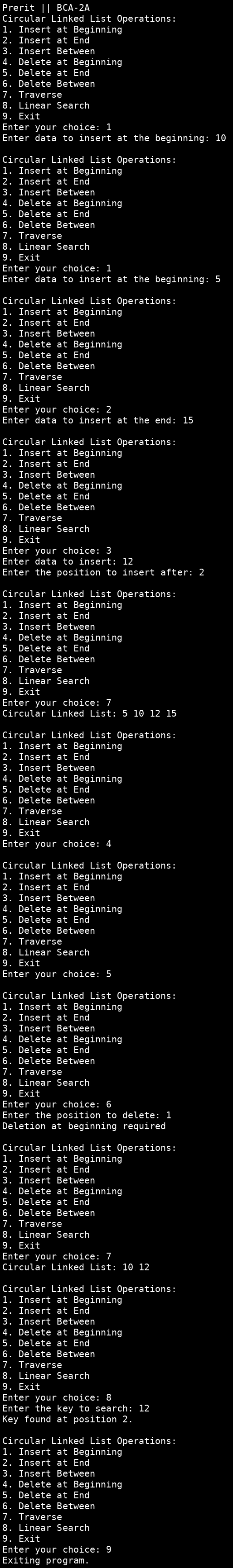
### Algorithm

```  
// Algorithm for Circular Linked List Operations  
  
// Structure Definition:  
// Node:  
// DATA: Integer  
// NEXT: Pointer to Node  
  
// Global Variable:  
// HEAD: Pointer to Node (initially NULL)  
  
// INSERT AT BEGINNING Algorithm:  
// INPUT: DATA (Integer)  
1. ALLOCATE memory for NEWNODE of type Node.  
2. SET NEWNODE→DATA = DATA.  
3. IF HEAD is NULL THEN  
4. SET NEWNODE→NEXT = NEWNODE.  
5. SET HEAD = NEWNODE.  
6. ELSE  
7. SET TEMP = HEAD.  
8. WHILE TEMP→NEXT != HEAD DO  
9. SET TEMP = TEMP→NEXT.  
10. [END OF LOOP]  
11. SET TEMP→NEXT = NEWNODE.  
12. SET NEWNODE→NEXT = HEAD.  
13. SET HEAD = NEWNODE.  
14. [END OF IF]  
15. EXIT  
  
// INSERT AT END Algorithm:  
// INPUT: DATA (Integer)  
1. ALLOCATE memory for NEWNODE of type Node.  
2. SET NEWNODE→DATA = DATA.  
3. IF HEAD is NULL THEN  
4. SET NEWNODE→NEXT = NEWNODE.  
5. SET HEAD = NEWNODE.  
6. ELSE  
7. SET TEMP = HEAD.  
8. WHILE TEMP→NEXT != HEAD DO  
9. SET TEMP = TEMP→NEXT.  
10. [END OF LOOP]  
11. SET TEMP→NEXT = NEWNODE.  
12. SET NEWNODE→NEXT = HEAD.  
13. [END OF IF]  
14. EXIT  
  
// INSERT BETWEEN Algorithm:  
// INPUT: DATA (Integer), POSITION (Integer)  
1. IF HEAD is NULL THEN  
2. PRINT "List is empty. Cannot insert between."  
3. EXIT  
4. [END OF IF]  
5. ALLOCATE memory for NEWNODE of type Node.  
6. SET NEWNODE→DATA = DATA.  
7. SET TEMP = HEAD.  
8. SET COUNT = 1.  
9. WHILE COUNT < POSITION AND TEMP→NEXT != HEAD DO  
10. SET TEMP = TEMP→NEXT.  
11. INCREMENT COUNT.  
12. [END OF LOOP]  
13. IF COUNT != POSITION THEN  
14. PRINT "Position not found."  
15. EXIT  
16. [END OF IF]  
17. SET NEWNODE→NEXT = TEMP→NEXT.  
18. SET TEMP→NEXT = NEWNODE.  
19. EXIT  
  
// DELETE AT BEGINNING Algorithm:  
1. IF HEAD is NULL THEN  
2. PRINT "List is empty. Cannot delete."  
3. EXIT  
4. [END OF IF]  
5. IF HEAD→NEXT == HEAD THEN  
6. FREE(HEAD).  
7. SET HEAD = NULL.  
8. EXIT  
9. [END OF IF]  
10. SET TEMP = HEAD.  
11. SET LAST = HEAD.  
12. WHILE LAST→NEXT != HEAD DO  
13. SET LAST = LAST→NEXT.  
14. [END OF LOOP]  
15. SET HEAD = HEAD→NEXT.  
16. SET LAST→NEXT = HEAD.  
17. FREE(TEMP).  
18. EXIT  
  
// DELETE AT END Algorithm:  
1. IF HEAD is NULL THEN  
2. PRINT "List is empty. Cannot delete."  
3. EXIT  
4. [END OF IF]  
5. IF HEAD→NEXT == HEAD THEN  
6. FREE(HEAD).  
7. SET HEAD = NULL.  
8. EXIT  
9. [END OF IF]  
10. SET TEMP = HEAD.  
11. SET PREV = NULL.  
12. WHILE TEMP→NEXT != HEAD DO  
13. SET PREV = TEMP.  
14. SET TEMP = TEMP→NEXT.  
15. [END OF LOOP]  
16. SET PREV→NEXT = HEAD.  
17. FREE(TEMP).  
18. EXIT  
  
// DELETE BETWEEN Algorithm:  
// INPUT: POSITION (Integer)  
1. IF HEAD is NULL THEN  
2. PRINT "List is empty. Cannot delete."  
3. EXIT  
4. [END OF IF]  
5. IF HEAD→NEXT == HEAD AND POSITION == 1 THEN  
6. FREE(HEAD).  
7. SET HEAD = NULL.  
8. EXIT  
9. [END OF IF]  
10. SET TEMP = HEAD.  
11. SET PREV = NULL.  
12. SET COUNT = 1.  
13. WHILE COUNT < POSITION AND TEMP→NEXT != HEAD DO  
14. SET PREV = TEMP.  
15. SET TEMP = TEMP→NEXT.  
16. INCREMENT COUNT.  
17. [END OF LOOP]  
18. IF COUNT != POSITION THEN  
19. PRINT "Position not found."  
20. EXIT  
21. [END OF IF]  
22. IF TEMP == HEAD THEN  
23. PRINT "Deletion at beginning required"  
24. EXIT  
25. [END OF IF]  
26. SET PREV→NEXT = TEMP→NEXT.  
27. FREE(TEMP).  
28. EXIT  
  
// TRAVERSE Algorithm:  
1. IF HEAD is NULL THEN  
2. PRINT "List is empty."  
3. EXIT  
4. [END OF IF]  
5. SET TEMP = HEAD.  
6. DO  
7. PRINT TEMP→DATA.  
8. SET TEMP = TEMP→NEXT.  
9. WHILE TEMP != HEAD.  
10. [END OF LOOP]  
11. PRINT newline.  
12. EXIT  
  
// LINEAR SEARCH Algorithm:  
// INPUT: KEY (Integer)  
1. IF HEAD is NULL THEN  
2. PRINT "List is empty."  
3. RETURN -1.  
4. [END OF IF]  
5. SET TEMP = HEAD.  
6. SET POSITION = 1.  
7. DO  
8. IF TEMP→DATA == KEY THEN  
9. RETURN POSITION.  
10. [END OF IF]  
11. SET TEMP = TEMP→NEXT.  
12. INCREMENT POSITION.  
13. WHILE TEMP != HEAD.  
14. [END OF LOOP]  
15. RETURN -1.  
16. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*next;  
};  
  
struct Node \*head = NULL;  
  
void insertBeginning(int data) {  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 if (head == NULL) {  
 newNode->next = newNode;  
 head = newNode;  
 } else {  
 struct Node \*temp = head;  
 while (temp->next != head) {  
 temp = temp->next;  
 }  
 temp->next = newNode;  
 newNode->next = head;  
 head = newNode;  
 }  
}  
  
void insertEnd(int data) {  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 if (head == NULL) {  
 newNode->next = newNode;  
 head = newNode;  
 } else {  
 struct Node \*temp = head;  
 while (temp->next != head) {  
 temp = temp->next;  
 }  
 temp->next = newNode;  
 newNode->next = head;  
 }  
}  
  
void insertBetween(int data, int position) {  
 if (head == NULL) {  
 printf("List is empty. Cannot insert between.\n");  
 return;  
 }  
 struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 struct Node \*temp = head;  
 int count = 1;  
 while (count < position && temp->next != head) {  
 temp = temp->next;  
 count++;  
 }  
 if (count != position) {  
 printf("Position not found.\n");  
 return;  
 }  
 newNode->next = temp->next;  
 temp->next = newNode;  
}  
  
void deleteBeginning() {  
 if (head == NULL) {  
 printf("List is empty. Cannot delete.\n");  
 return;  
 }  
 if (head->next == head) {  
 free(head);  
 head = NULL;  
 return;  
 }  
 struct Node \*temp = head;  
 struct Node \*last = head;  
 while (last->next != head) {  
 last = last->next;  
 }  
 head = head->next;  
 last->next = head;  
 free(temp);  
}  
  
void deleteEnd() {  
 if (head == NULL) {  
 printf("List is empty. Cannot delete.\n");  
 return;  
 }  
 if (head->next == head) {  
 free(head);  
 head = NULL;  
 return;  
 }  
 struct Node \*temp = head;  
 struct Node \*prev = NULL;  
 while (temp->next != head) {  
 prev = temp;  
 temp = temp->next;  
 }  
 prev->next = head;  
 free(temp);  
}  
  
void deleteBetween(int position) {  
 if (head == NULL) {  
 printf("List is empty. Cannot delete.\n");  
 return;  
 }  
 if (head->next == head && position == 1) {  
 free(head);  
 head = NULL;  
 return;  
 }  
  
 struct Node \*temp = head;  
 struct Node \*prev = NULL;  
 int count = 1;  
  
 while (count < position && temp->next != head) {  
 prev = temp;  
 temp = temp->next;  
 count++;  
 }  
  
 if (count != position) {  
 printf("Position not found.\n");  
 return;  
 }  
  
 if (temp == head) {  
 printf("Deletion at beginning required\n");  
 return;  
 }  
 prev->next = temp->next;  
 free(temp);  
}  
  
  
void traverse() {  
 if (head == NULL) {  
 printf("List is empty.\n");  
 return;  
 }  
 struct Node \*temp = head;  
 do {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 } while (temp != head);  
 printf("\n");  
}  
  
int linearSearch(int key) {  
 if (head == NULL) {  
 printf("List is empty.\n");  
 return -1;  
 }  
 struct Node \*temp = head;  
 int position = 1;  
 do {  
 if (temp->data == key) {  
 return position;  
 }  
 temp = temp->next;  
 position++;  
 } while (temp != head);  
 return -1;  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, data, position, key, searchResult;  
  
 do {  
 printf("\nCircular Linked List Operations:\n");  
 printf("1. Insert at Beginning\n");  
 printf("2. Insert at End\n");  
 printf("3. Insert Between\n");  
 printf("4. Delete at Beginning\n");  
 printf("5. Delete at End\n");  
 printf("6. Delete Between\n");  
 printf("7. Traverse\n");  
 printf("8. Linear Search\n");  
 printf("9. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to insert at the beginning: ");  
 scanf("%d", &data);  
 insertBeginning(data);  
 break;  
 case 2:  
 printf("Enter data to insert at the end: ");  
 scanf("%d", &data);  
 insertEnd(data);  
 break;  
 case 3:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 printf("Enter the position to insert after: ");  
 scanf("%d", &position);  
 insertBetween(data, position);  
 break;  
 case 4:  
 deleteBeginning();  
 break;  
 case 5:  
 deleteEnd();  
 break;  
 case 6:  
 printf("Enter the position to delete: ");  
 scanf("%d", &position);  
 deleteBetween(position);  
 break;  
 case 7:  
 printf("Circular Linked List: ");  
 traverse();  
 break;  
 case 8:  
 printf("Enter the key to search: ");  
 scanf("%d", &key);  
 searchResult = linearSearch(key);  
 if (searchResult == -1) {  
 printf("Key not found.\n");  
 } else {  
 printf("Key found at position %d.\n", searchResult);  
 }  
 break;  
 case 9:  
 printf("Exiting program.\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 9);  
  
 return 0;  
}

### Output:



## 29. WAP to perform polynomial addition using linked lists

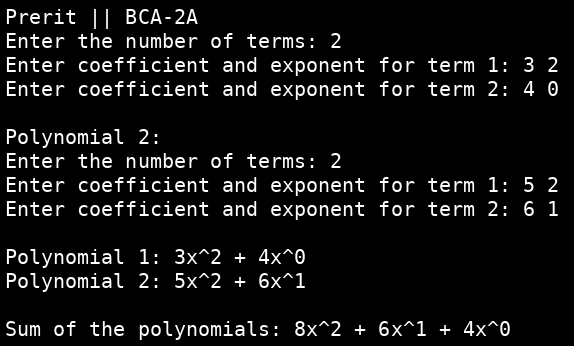
### Algorithm

```  
// Algorithm for Polynomial Addition  
  
// Sub-algorithm: CREATE\_TERM(coefficient, exponent)  
1. ALLOCATE memory for NEW\_TERM.  
2. SET NEW\_TERM→COEFFICIENT = coefficient.  
3. SET NEW\_TERM→EXPONENT = exponent.  
4. SET NEW\_TERM→NEXT = NULL.  
5. RETURN NEW\_TERM.  
  
// Sub-algorithm: INSERT\_TERM(POLY, coefficient, exponent)  
1. CALL CREATE\_TERM(coefficient, exponent) to get NEW\_TERM.  
2. IF POLY is NULL THEN  
 SET POLY = NEW\_TERM.  
 EXIT.  
 [END OF IF]  
3. IF exponent > POLY→EXPONENT THEN  
 SET NEW\_TERM→NEXT = POLY.  
 SET POLY = NEW\_TERM.  
 EXIT.  
 [END OF IF]  
4. SET CURR = POLY.  
5. SET PREV = NULL.  
6. WHILE CURR is not NULL AND CURR→EXPONENT > exponent DO  
 SET PREV = CURR.  
 SET CURR = CURR→NEXT.  
 [END OF LOOP]  
7. IF CURR is not NULL AND CURR→EXPONENT == exponent THEN  
 EXIT.  
 [END OF IF]  
8. SET NEW\_TERM→NEXT = CURR.  
9. IF PREV is not NULL THEN  
 SET PREV→NEXT = NEW\_TERM.  
 ELSE  
 SET POLY = NEW\_TERM.  
 [END OF IF]  
  
// Sub-algorithm: CREATE\_POLYNOMIAL()  
1. INPUT N (number of terms).  
2. SET POLY = NULL.  
3. FOR I = 0 TO N-1 DO  
 INPUT COEFFICIENT, EXPONENT.  
 CALL INSERT\_TERM(POLY, COEFFICIENT, EXPONENT).  
 [END OF LOOP]  
4. RETURN POLY.  
  
// Sub-algorithm: DISPLAY\_POLYNOMIAL(POLY)  
1. IF POLY is NULL THEN  
 OUTPUT "0".  
 EXIT.  
 [END OF IF]  
2. SET CURR = POLY.  
3. WHILE CURR is not NULL DO  
 OUTPUT CURR→COEFFICIENT, "x^", CURR→EXPONENT.  
 SET CURR = CURR→NEXT.  
 IF CURR is not NULL THEN  
 OUTPUT " + ".  
 [END OF IF]  
 [END OF LOOP]  
  
// Sub-algorithm: ADD\_POLYNOMIALS(POLY1, POLY2)  
1. SET RESULT = NULL.  
2. SET CURR1 = POLY1.  
3. SET CURR2 = POLY2.  
4. WHILE CURR1 is not NULL AND CURR2 is not NULL DO  
 IF CURR1→EXPONENT == CURR2→EXPONENT THEN  
 SET SUM = CURR1→COEFFICIENT + CURR2→COEFFICIENT.  
 IF SUM is not equal to 0 THEN  
 CALL INSERT\_TERM(RESULT, SUM, CURR1→EXPONENT).  
 [END OF IF]  
 SET CURR1 = CURR1→NEXT.  
 SET CURR2 = CURR2→NEXT.  
 ELSE IF CURR1→EXPONENT > CURR2→EXPONENT THEN  
 CALL INSERT\_TERM(RESULT, CURR1→COEFFICIENT, CURR1→EXPONENT).  
 SET CURR1 = CURR1→NEXT.  
 ELSE  
 CALL INSERT\_TERM(RESULT, CURR2→COEFFICIENT, CURR2→EXPONENT).  
 SET CURR2 = CURR2→NEXT.  
 [END OF IF]  
 [END OF LOOP]  
5. WHILE CURR1 is not NULL DO  
 CALL INSERT\_TERM(RESULT, CURR1→COEFFICIENT, CURR1→EXPONENT).  
 SET CURR1 = CURR1→NEXT.  
 [END OF LOOP]  
6. WHILE CURR2 is not NULL DO  
 CALL INSERT\_TERM(RESULT, CURR2→COEFFICIENT, CURR2→EXPONENT).  
 SET CURR2 = CURR2→NEXT.  
 [END OF LOOP]  
7. RETURN RESULT.  
  
// Sub-algorithm: FREE\_POLYNOMIAL(POLY)  
1. SET CURR = POLY.  
2. WHILE CURR is not NULL DO  
 SET TEMP = CURR.  
 SET CURR = CURR→NEXT.  
 FREE(TEMP).  
 [END OF LOOP]  
  
// Main Algorithm  
1. CALL CREATE\_POLYNOMIAL() to create POLY1.  
2. CALL CREATE\_POLYNOMIAL() to create POLY2.  
3. CALL DISPLAY\_POLYNOMIAL(POLY1).  
4. CALL DISPLAY\_POLYNOMIAL(POLY2).  
5. CALL ADD\_POLYNOMIALS(POLY1, POLY2) to get SUM.  
6. CALL DISPLAY\_POLYNOMIAL(SUM).  
7. CALL FREE\_POLYNOMIAL(POLY1).  
8. CALL FREE\_POLYNOMIAL(POLY2).  
9. CALL FREE\_POLYNOMIAL(SUM).  
10. EXIT.  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Term {  
 int coefficient;  
 int exponent;  
 struct Term \*next;  
};  
  
struct Term \*createTerm(int coefficient, int exponent) {  
 struct Term \*newTerm = (struct Term \*)malloc(sizeof(struct Term));  
 newTerm->coefficient = coefficient;  
 newTerm->exponent = exponent;  
 newTerm->next = NULL;  
 return newTerm;  
}  
  
void insertTerm(struct Term \*\*poly, int coefficient, int exponent) {  
 struct Term \*newTerm = createTerm(coefficient, exponent);  
 if (\*poly == NULL) {  
 \*poly = newTerm;  
 return;  
 }  
  
 if (exponent > (\*poly)->exponent) {  
 newTerm->next = \*poly;  
 \*poly = newTerm;  
 return;  
 }  
  
 struct Term \*curr = \*poly;  
 struct Term \*prev = NULL;  
  
 while (curr != NULL && curr->exponent > exponent) {  
 prev = curr;  
 curr = curr->next;  
 }  
  
 if (curr != NULL && curr->exponent == exponent) return;  
  
 newTerm->next = curr;  
 if(prev != NULL)  
 prev->next = newTerm;  
 else  
 \*poly = newTerm;  
}  
  
struct Term \*createPolynomial() {  
 int n, coefficient, exponent;  
 struct Term \*poly = NULL;  
 printf("Enter the number of terms: ");  
 scanf("%d", &n);  
 for (int i = 0; i < n; i++) {  
 printf("Enter coefficient and exponent for term %d: ", i + 1);  
 scanf("%d %d", &coefficient, &exponent);  
 insertTerm(&poly, coefficient, exponent);  
 }  
 return poly;  
}  
  
void displayPolynomial(struct Term \*poly) {  
 if (poly == NULL) {  
 printf("0\n");  
 return;  
 }  
  
 struct Term \*curr = poly;  
 while (curr != NULL) {  
 printf("%dx^%d", curr->coefficient, curr->exponent);  
 curr = curr->next;  
 if (curr != NULL) {  
 printf(" + ");  
 }  
 }  
 printf("\n");  
}  
  
struct Term \*addPolynomials(struct Term \*poly1, struct Term \*poly2) {  
 struct Term \*result = NULL;  
 struct Term \*curr1 = poly1;  
 struct Term \*curr2 = poly2;  
  
 while (curr1 != NULL && curr2 != NULL) {  
 if (curr1->exponent == curr2->exponent) {  
 int sum = curr1->coefficient + curr2->coefficient;  
 if (sum != 0) {  
 insertTerm(&result, sum, curr1->exponent);  
 }  
 curr1 = curr1->next;  
 curr2 = curr2->next;  
 } else if (curr1->exponent > curr2->exponent) {  
 insertTerm(&result, curr1->coefficient, curr1->exponent);  
 curr1 = curr1->next;  
 } else {  
 insertTerm(&result, curr2->coefficient, curr2->exponent);  
 curr2 = curr2->next;  
 }  
 }  
  
 while (curr1 != NULL) {  
 insertTerm(&result, curr1->coefficient, curr1->exponent);  
 curr1 = curr1->next;  
 }  
  
 while (curr2 != NULL) {  
 insertTerm(&result, curr2->coefficient, curr2->exponent);  
 curr2 = curr2->next;  
 }  
  
 return result;  
}  
  
void freePolynomial(struct Term \*poly) {  
 struct Term \*curr = poly;  
 while (curr != NULL) {  
 struct Term \*temp = curr;  
 curr = curr->next;  
 free(temp);  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 struct Term \*poly1, \*poly2, \*sum;  
  
 printf("\nPolynomial 1:\n");  
 poly1 = createPolynomial();  
  
 printf("\nPolynomial 2:\n");  
 poly2 = createPolynomial();  
  
 printf("\nPolynomial 1: ");  
 displayPolynomial(poly1);  
  
 printf("Polynomial 2: ");  
 displayPolynomial(poly2);  
  
 sum = addPolynomials(poly1, poly2);  
  
 printf("\nSum of the polynomials: ");  
 displayPolynomial(sum);  
  
 freePolynomial(poly1);  
 freePolynomial(poly2);  
 freePolynomial(sum);  
  
 return 0;  
}

### Output:



## 30. Write a Program to Perform Pop, Push, Traverse operations on the stack using array (Static Stack).

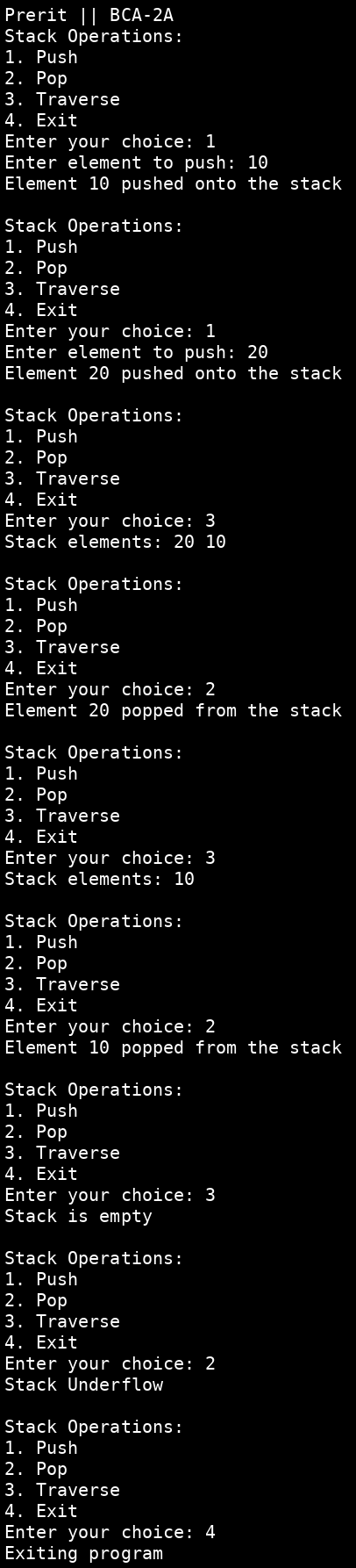
### Algorithm

```  
// Algorithm for Stack Operations (Push, Pop, Traverse)  
  
// Initialization  
1. SET MAX\_SIZE = 5  
2. CREATE stack[MAX\_SIZE]  
3. SET top = -1  
  
// Push Operation  
4. PUSH(element):  
5. IF top == MAX\_SIZE - 1 THEN  
6. PRINT "Stack Overflow"  
7. ELSE  
8. top = top + 1  
9. stack[top] = element  
10. PRINT "Element" element "pushed onto the stack"  
 [END OF IF]  
11. RETURN  
  
// Pop Operation  
12. POP():  
13. IF top == -1 THEN  
14. PRINT "Stack Underflow"  
15. ELSE  
16. PRINT "Element" stack[top] "popped from the stack"  
17. top = top - 1  
 [END OF IF]  
18. RETURN  
  
// Traverse Operation  
19. TRAVERSE():  
20. IF top == -1 THEN  
21. PRINT "Stack is empty"  
22. ELSE  
23. PRINT "Stack elements: "  
24. FOR i = top DOWNTO 0  
25. PRINT stack[i] " "  
 [END OF LOOP]  
26. PRINT newline  
 [END OF IF]  
27. RETURN  
  
// Main Program  
28. MAIN():  
29. DO  
30. PRINT "Stack Operations Menu"  
31. INPUT choice  
32. SWITCH choice:  
33. CASE 1: // Push  
34. INPUT element  
35. PUSH(element)  
36. BREAK  
37. CASE 2: // Pop  
38. POP()  
39. BREAK  
40. CASE 3: // Traverse  
41. TRAVERSE()  
42. BREAK  
43. CASE 4: // Exit  
44. PRINT "Exiting program"  
45. BREAK  
46. DEFAULT:  
47. PRINT "Invalid choice"  
 [END OF SWITCH]  
48. WHILE choice != 4  
49. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
#define MAX\_SIZE 5  
  
int stack[MAX\_SIZE];  
int top = -1;  
  
void push(int element) {  
 if (top == MAX\_SIZE - 1) {  
 printf("Stack Overflow\n");  
 } else {  
 top++;  
 stack[top] = element;  
 printf("Element %d pushed onto the stack\n", element);  
 }  
}  
  
void pop() {  
 if (top == -1) {  
 printf("Stack Underflow\n");  
 } else {  
 printf("Element %d popped from the stack\n", stack[top]);  
 top--;  
 }  
}  
  
void traverse() {  
 if (top == -1) {  
 printf("Stack is empty\n");  
 } else {  
 printf("Stack elements: ");  
 for (int i = top; i >= 0; i--) {  
 printf("%d ", stack[i]);  
 }  
 printf("\n");  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, element;  
  
 do {  
 printf("\nStack Operations:\n");  
 printf("1. Push\n");  
 printf("2. Pop\n");  
 printf("3. Traverse\n");  
 printf("4. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter element to push: ");  
 scanf("%d", &element);  
 push(element);  
 break;  
 case 2:  
 pop();  
 break;  
 case 3:  
 traverse();  
 break;  
 case 4:  
 printf("Exiting program\n");  
 break;  
 default:  
 printf("Invalid choice\n");  
 }  
 } while (choice != 4);  
  
 return 0;  
}

### Output:



## 31. (Optional) WAP to Convert Infix Expression to Postfix form using Stack.

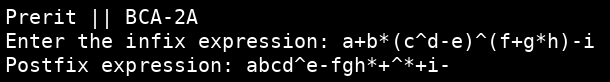
### Algorithm

1. \*\*ALGORITHM\*\* infixToPostfix(infix, postfix)  
2. SET STACK as an array of characters.  
3. SET TOP = -1.  
4. SET i = 0, j = 0.  
5. WHILE infix\[i] != '\0' DO  
 \* IF infix\[i] is alphanumeric THEN  
 \* SET postfix\[j++] = infix\[i].  
 [END OF IF]  
 \* ELSE IF infix\[i] == '(' THEN  
 \* SET STACK\[++TOP] = infix\[i].  
 [END OF IF]  
 \* ELSE IF infix\[i] == ')' THEN  
 \* WHILE TOP >= 0 AND STACK\[TOP] != '(' DO  
 \* SET postfix\[j++] = STACK\[TOP--].  
 [END OF LOOP]  
 \* IF TOP >= 0 AND STACK\[TOP] == '(' THEN  
 \* DECREMENT TOP.  
 [END OF IF]  
 [END OF IF]  
 \* ELSE  
 \* WHILE TOP >= 0 AND precedence(STACK\[TOP]) >= precedence(infix\[i]) DO  
 \* SET postfix\[j++] = STACK\[TOP--].  
 [END OF LOOP]  
 \* SET STACK\[++TOP] = infix\[i].  
 [END OF ELSE]  
 \* INCREMENT i.  
 [END OF LOOP]  
6. WHILE TOP >= 0 DO  
 \* SET postfix\[j++] = STACK\[TOP--].  
 [END OF LOOP]  
7. SET postfix\[j] = '\0'.  
8. EXIT.

### Code:

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <ctype.h>  
  
int precedence(char op) {  
 switch (op) {  
 case '^': return 3;  
 case '\*':  
 case '/': return 2;  
 case '+':  
 case '-': return 1;  
 default: return 0;  
 }  
}  
  
void infixToPostfix(char infix[], char postfix[]) {  
 char stack[100];  
 int top = -1;  
 int i, j;  
 for (i = 0, j = 0; infix[i] != '\0'; i++) {  
 if (isalnum(infix[i])) {  
 postfix[j++] = infix[i];  
 } else if (infix[i] == '(') {  
 stack[++top] = infix[i];  
 } else if (infix[i] == ')') {  
 while (top >= 0 && stack[top] != '(') {  
 postfix[j++] = stack[top--];  
 }  
 if (top >= 0 && stack[top] == '(') {  
 top--;  
 }  
 } else {  
 while (top >= 0 && precedence(stack[top]) >= precedence(infix[i])) {  
 postfix[j++] = stack[top--];  
 }  
 stack[++top] = infix[i];  
 }  
 }  
 while (top >= 0) {  
 postfix[j++] = stack[top--];  
 }  
 postfix[j] = '\0';  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 char infix[100], postfix[100];  
 printf("\nEnter the infix expression: ");  
 scanf("%s", infix);  
 infixToPostfix(infix, postfix);  
 printf("Postfix expression: %s\n", postfix);  
 return 0;  
}

### Output:



## 32. (Optional) WAP to Convert Infix Expression to Prefix form using Stack.

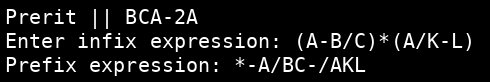
### Algorithm

```  
ALGORITHM infixToPrefix  
  
INPUT: Infix expression (infix)  
OUTPUT: Prefix expression (prefix)  
  
Step 1: SET len = length of infix  
Step 2: ALLOCATE stack of size len  
Step 3: SET top = -1  
Step 4: ALLOCATE reversedInfix of size len + 1  
Step 5: FOR i = 0 to len - 1 DO  
 SET reversedInfix[i] = infix[len - 1 - i]  
 IF reversedInfix[i] == '(' THEN  
 SET reversedInfix[i] = ')'  
 ELSE IF reversedInfix[i] == ')' THEN  
 SET reversedInfix[i] = '('  
 [END OF IF]  
[END OF LOOP]  
Step 6: SET reversedInfix[len] = '\0'  
Step 7: SET j = 0  
Step 8: FOR i = 0 to length of reversedInfix -1 DO  
 IF reversedInfix[i] is alphanumeric THEN  
 SET prefix[j] = reversedInfix[i]  
 SET j = j + 1  
 ELSE IF reversedInfix[i] == '(' THEN  
 SET top = top + 1  
 SET stack[top] = reversedInfix[i]  
 ELSE IF reversedInfix[i] == ')' THEN  
 WHILE top >= 0 AND stack[top] != '(' DO  
 SET prefix[j] = stack[top]  
 SET top = top - 1  
 SET j = j + 1  
 [END OF LOOP]  
 IF top >= 0 AND stack[top] == '(' THEN  
 SET top = top - 1  
 [END OF IF]  
 ELSE  
 WHILE top >= 0 AND precedence(reversedInfix[i]) <= precedence(stack[top]) DO  
 SET prefix[j] = stack[top]  
 SET top = top - 1  
 SET j = j + 1  
 [END OF LOOP]  
 SET top = top + 1  
 SET stack[top] = reversedInfix[i]  
 [END OF IF]  
[END OF LOOP]  
Step 9: WHILE top >= 0 DO  
 SET prefix[j] = stack[top]  
 SET top = top - 1  
 SET j = j + 1  
[END OF LOOP]  
Step 10: SET prefix[j] = '\0'  
Step 11: ALLOCATE tempPrefix of size j + 1  
Step 12: FOR i = 0 to j - 1 DO  
 SET tempPrefix[i] = prefix[j - 1 - i]  
[END OF LOOP]  
Step 13: SET tempPrefix[j] = '\0'  
Step 14: COPY tempPrefix to prefix  
Step 15: EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <ctype.h>  
  
int precedence(char operator) {  
 switch (operator) {  
 case '+':  
 case '-':  
 return 1;  
 case '\*':  
 case '/':  
 return 2;  
 case '^':  
 return 3;  
 default:  
 return 0;  
 }  
}  
  
void infixToPrefix(char \*infix, char \*prefix) {  
 int i, j = 0;  
 int len = strlen(infix);  
 char stack[len];  
 int top = -1;  
  
 char reversedInfix[len + 1];  
 for (i = 0; i < len; i++) {  
 reversedInfix[i] = infix[len - 1 - i];  
 if (reversedInfix[i] == '(')  
 reversedInfix[i] = ')';  
 else if (reversedInfix[i] == ')')  
 reversedInfix[i] = '(';  
 }  
 reversedInfix[len] = '\0';  
  
 for (i = 0; reversedInfix[i] != '\0'; i++) {  
 if (isalnum(reversedInfix[i])) {  
 prefix[j++] = reversedInfix[i];  
 } else if (reversedInfix[i] == '(') {  
 stack[++top] = reversedInfix[i];  
 } else if (reversedInfix[i] == ')') {  
 while (top >= 0 && stack[top] != '(') {  
 prefix[j++] = stack[top--];  
 }  
 if (top >= 0 && stack[top] == '(')  
 top--;  
 } else {  
 while (top >= 0 && precedence(reversedInfix[i]) <= precedence(stack[top])) {  
 prefix[j++] = stack[top--];  
 }  
 stack[++top] = reversedInfix[i];  
 }  
 }  
  
 while (top >= 0) {  
 prefix[j++] = stack[top--];  
 }  
  
 prefix[j] = '\0';  
  
 char tempPrefix[j + 1];  
 for (i = 0; i < j; i++) {  
 tempPrefix[i] = prefix[j - 1 - i];  
 }  
 tempPrefix[j] = '\0';  
  
 strcpy(prefix, tempPrefix);  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 char infix[100];  
 char prefix[100];  
  
 printf("\nEnter infix expression: ");  
 scanf("%s", infix);  
  
 infixToPrefix(infix, prefix);  
  
 printf("Prefix expression: %s\n", prefix);  
  
 return 0;  
}

### Output:



## 33. WAP to evaluate Postfix expression using Stack.

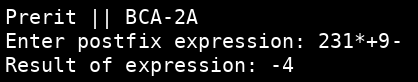
### Algorithm

```  
Algorithm: Postfix Expression Evaluation  
  
1. INITIALIZE: SET top = -1 (stack pointer)  
  
2. INPUT: Read postfix expression into string `expression`  
  
3. SET e = `expression` (pointer to the start of the expression)  
  
4. WHILE (\*e != '\0') DO  
 5. IF (isdigit(\*e)) THEN  
 6. SET num = \*e - 48 (convert character to integer)  
 7. CALL push(num)  
 [END OF IF]  
 8. ELSE  
 9. SET n1 = pop()  
 10. SET n2 = pop()  
 11. SWITCH (\*e) DO  
 12. CASE '+': SET result = n2 + n1  
 13. CASE '-': SET result = n2 - n1  
 14. CASE '\*': SET result = n2 \* n1  
 15. CASE '/': SET result = n2 / n1  
 [END OF SWITCH]  
 16. CALL push(result)  
 [END OF ELSE]  
 17. SET e = e + 1 (move to the next character)  
 [END OF LOOP]  
  
18. SET result = pop()  
  
19. OUTPUT: Print `result`  
  
20. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
  
int stack[100];  
int top = -1;  
  
void push(int item) {  
 stack[++top] = item;  
}  
  
int pop() {  
 return stack[top--];  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 char expression[100];  
 char \*e;  
 int n1, n2, num, result;  
  
 printf("Enter postfix expression: ");  
 scanf("%s", expression);  
  
 e = expression;  
  
 while (\*e != '\0') {  
 if (isdigit(\*e)) {  
 num = \*e - 48;  
 push(num);  
 } else {  
 n1 = pop();  
 n2 = pop();  
 switch (\*e) {  
 case '+':  
 result = n2 + n1;  
 break;  
 case '-':  
 result = n2 - n1;  
 break;  
 case '\*':  
 result = n2 \* n1;  
 break;  
 case '/':  
 result = n2 / n1;  
 break;  
 }  
 push(result);  
 }  
 e++;  
 }  
 printf("Result of expression: %d\n", pop());  
 return 0;  
}

### Output:



## 34. WAP to reverse the String using Stack

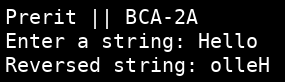
### Algorithm

```  
// Algorithm: String Reversal using Stack  
  
Step 1: INITIALIZE string `str` of size 100.  
  
Step 2: INPUT string `str` from user.  
  
Step 3: SET `n` = length of `str`.  
  
Step 4: CREATE stack `stack` of capacity `n`.  
  
Step 5: LOOP `i` from 0 to `n-1`  
 Step 6: PUSH `str[i]` onto `stack`.  
[END OF LOOP]  
  
Step 7: PRINT "Reversed string: ".  
  
Step 8: WHILE stack `stack` is not empty  
 Step 9: PRINT element popped from `stack`.  
[END OF LOOP]  
  
Step 10: PRINT newline character.  
  
Step 11: FREE memory allocated for `stack`→`array`.  
  
Step 12: FREE memory allocated for `stack`.  
  
Step 13: EXIT.  
```

### Code:

#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
struct Stack {  
 int top;  
 unsigned capacity;  
 char \*array;  
};  
  
struct Stack\* createStack(unsigned capacity) {  
 struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));  
 stack->capacity = capacity;  
 stack->top = -1;  
 stack->array = (char\*) malloc(stack->capacity \* sizeof(char));  
 return stack;  
}  
  
int isFull(struct Stack\* stack) {  
 return stack->top == stack->capacity - 1;  
}  
  
int isEmpty(struct Stack\* stack) {  
 return stack->top == -1;  
}  
  
void push(struct Stack\* stack, char item) {  
 if (isFull(stack))  
 return;  
 stack->array[++stack->top] = item;  
}  
  
char pop(struct Stack\* stack) {  
 if (isEmpty(stack))  
 return '\0';  
 return stack->array[stack->top--];  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 char str[100];  
 printf("\nEnter a string: ");  
 scanf("%s", str);  
  
 int n = strlen(str);  
 struct Stack\* stack = createStack(n);  
  
 for (int i = 0; i < n; i++) {  
 push(stack, str[i]);  
 }  
  
 printf("Reversed string: ");  
 while (!isEmpty(stack)) {  
 printf("%c", pop(stack));  
 }  
 printf("\n");  
 free(stack->array);  
 free(stack);  
  
 return 0;  
}

### Output:



## 35. WAP to perform different operations with Queue such as Insert, Delete, Display of elements using array. (Linear Queue or Static Queue)

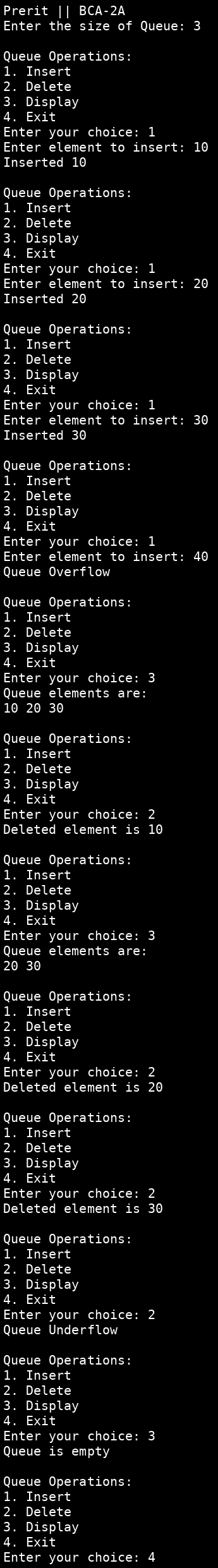
### Algorithm

```  
// Algorithm for Queue Operations (Array Implementation)  
  
// Global Variables:  
// QUEUE: Array to store queue elements.  
// FRONT: Index of the front element (-1 if empty).  
// REAR: Index of the rear element (-1 if empty).  
// SIZE: Maximum size of the queue.  
  
// Initialization  
1. START  
  
// Main Function  
2. PRINT "Enter the size of Queue: "  
3. READ SIZE  
4. WHILE (TRUE)  
5. PRINT "Queue Operations:\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice: "  
6. READ CHOICE  
7. SWITCH (CHOICE)  
 CASE 1: // Insert Operation  
8. PRINT "Enter element to insert: "  
9. READ ELEMENT  
10. CALL INSERT(ELEMENT)  
 BREAK  
 CASE 2: // Delete Operation  
11. CALL DELETE\_ELEMENT()  
 BREAK  
 CASE 3: // Display Operation  
12. CALL DISPLAY()  
 BREAK  
 CASE 4: // Exit  
13. EXIT  
 BREAK  
 DEFAULT:  
14. PRINT "Invalid choice"  
 BREAK  
 [END OF SWITCH]  
 [END OF LOOP]  
  
// Insert Function: INSERT(ELEMENT)  
15. IF REAR == SIZE - 1 THEN  
16. PRINT "Queue Overflow"  
17. RETURN  
 [END OF IF]  
18. IF FRONT == -1 THEN  
19. SET FRONT = 0  
 [END OF IF]  
20. SET REAR = REAR + 1  
21. SET QUEUE[REAR] = ELEMENT  
22. PRINT "Inserted ", ELEMENT  
  
// Delete Function: DELETE\_ELEMENT()  
23. IF FRONT == -1 OR FRONT > REAR THEN  
24. PRINT "Queue Underflow"  
25. RETURN  
 [END OF IF]  
26. PRINT "Deleted element is ", QUEUE[FRONT]  
27. SET FRONT = FRONT + 1  
28. IF FRONT > REAR THEN  
29. SET FRONT = -1  
30. SET REAR = -1  
 [END OF IF]  
  
// Display Function: DISPLAY()  
31. IF FRONT == -1 OR FRONT > REAR THEN  
32. PRINT "Queue is empty"  
33. RETURN  
 [END OF IF]  
34. PRINT "Queue elements are:"  
35. FOR I = FRONT TO REAR  
36. PRINT QUEUE[I], " "  
 [END OF LOOP]  
37. PRINT "\n"  
  
38. END  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int queue[100];  
int front = -1;  
int rear = -1;  
int size;  
  
void insert(int element) {  
 if (rear == size - 1) {  
 printf("Queue Overflow\n");  
 return;  
 }  
 if (front == -1)  
 front = 0;  
 rear++;  
 queue[rear] = element;  
 printf("Inserted %d\n", element);  
}  
  
void delete\_element() {  
 if (front == -1 || front > rear) {  
 printf("Queue Underflow\n");  
 return;  
 }  
 printf("Deleted element is %d\n", queue[front]);  
 front++;  
 if (front > rear) {  
 front = -1;  
 rear = -1;  
 }  
}  
  
void display() {  
 int i;  
 if (front == -1 || front > rear) {  
 printf("Queue is empty\n");  
 return;  
 }  
 printf("Queue elements are:\n");  
 for (i = front; i <= rear; i++)  
 printf("%d ", queue[i]);  
 printf("\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, element;  
 printf("\nEnter the size of Queue: ");  
 scanf("%d", &size);  
 while (1) {  
 printf("\nQueue Operations:\n");  
 printf("1. Insert\n");  
 printf("2. Delete\n");  
 printf("3. Display\n");  
 printf("4. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
 switch (choice) {  
 case 1:  
 printf("Enter element to insert: ");  
 scanf("%d", &element);  
 insert(element);  
 break;  
 case 2:  
 delete\_element();  
 break;  
 case 3:  
 display();  
 break;  
 case 4:  
 exit(0);  
 default:  
 printf("Invalid choice\n");  
 }  
 }  
 return 0;  
}

### Output:



## 36. WAP to perform different operations with Queue such as Insert, Delete, Display of elements using Circular Queue.

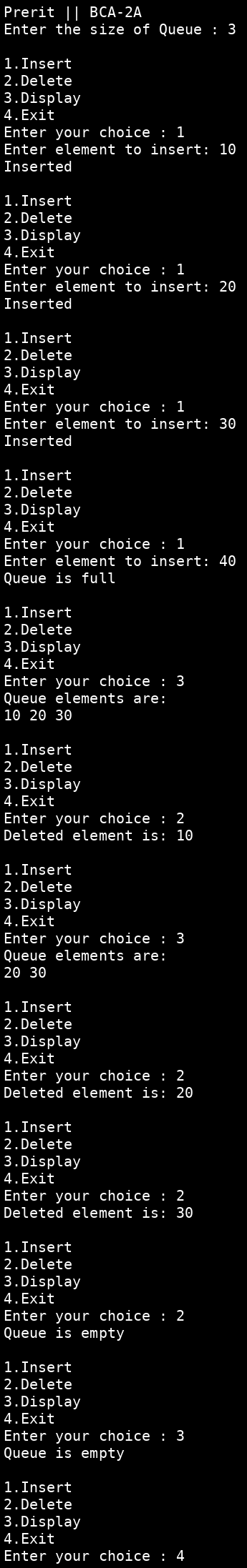
### Algorithm

```  
// Algorithm for Circular Queue Operations  
  
// Initialization:  
1. SET FRONT = -1  
2. SET REAR = -1  
3. INPUT SIZE  
  
// Main Loop:  
4. WHILE (TRUE) DO  
 5. DISPLAY Menu (1. Insert, 2. Delete, 3. Display, 4. Exit)  
 6. INPUT CHOICE  
 7. SWITCH (CHOICE)  
 8. CASE 1: // Insert  
 9. IF ((REAR + 1) MOD SIZE) == FRONT THEN  
 10. PRINT "Queue is full"  
 11. ELSE  
 12. INPUT VAL  
 13. IF FRONT == -1 THEN  
 14. SET FRONT = 0  
 [END OF IF]  
 15. SET REAR = (REAR + 1) MOD SIZE  
 16. SET QUEUE[REAR] = VAL  
 17. PRINT "Inserted"  
 [END OF IF]  
 18. BREAK  
 19. CASE 2: // Delete  
 20. IF FRONT == -1 THEN  
 21. PRINT "Queue is empty"  
 22. ELSE  
 23. PRINT "Deleted element is: ", QUEUE[FRONT]  
 24. IF FRONT == REAR THEN  
 25. SET FRONT = -1  
 26. SET REAR = -1  
 [END OF IF]  
 27. ELSE  
 28. SET FRONT = (FRONT + 1) MOD SIZE  
 [END OF ELSE]  
 [END OF IF]  
 29. BREAK  
 30. CASE 3: // Display  
 31. IF FRONT == -1 THEN  
 32. PRINT "Queue is empty"  
 33. ELSE  
 34. PRINT "Queue elements are:"  
 35. SET I = FRONT  
 36. WHILE I != REAR DO  
 37. PRINT QUEUE[I]  
 38. SET I = (I + 1) MOD SIZE  
 [END OF LOOP]  
 39. PRINT QUEUE[REAR]  
 [END OF IF]  
 40. BREAK  
 41. CASE 4: // Exit  
 42. EXIT  
 43. DEFAULT:  
 44. PRINT "Invalid choice"  
 45. BREAK  
 [END OF SWITCH]  
[END OF LOOP]  
  
//Exit  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
int queue[100];  
int front = -1, rear = -1, size;  
  
void insert() {  
 int val;  
 if ((rear + 1) % size == front) {  
 printf("Queue is full\n");  
 return;  
 }  
 printf("Enter element to insert: ");  
 scanf("%d", &val);  
 if (front == -1)  
 front = 0;  
 rear = (rear + 1) % size;  
 queue[rear] = val;  
 printf("Inserted\n");  
}  
  
void delete() {  
 if (front == -1) {  
 printf("Queue is empty\n");  
 return;  
 }  
 printf("Deleted element is: %d\n", queue[front]);  
 if (front == rear) {  
 front = -1;  
 rear = -1;  
 } else {  
 front = (front + 1) % size;  
 }  
}  
  
void display() {  
 int i;  
 if (front == -1) {  
 printf("Queue is empty\n");  
 return;  
 }  
 printf("Queue elements are: \n");  
 for (i = front; i != rear; i = (i + 1) % size) {  
 printf("%d ", queue[i]);  
 }  
 printf("%d ", queue[rear]);  
 printf("\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice;  
 printf("\nEnter the size of Queue : ");  
 scanf("%d", &size);  
 while (1) {  
 printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");  
 printf("Enter your choice : ");  
 scanf("%d", &choice);  
 switch (choice) {  
 case 1:  
 insert();  
 break;  
 case 2:  
 delete();  
 break;  
 case 3:  
 display();  
 break;  
 case 4:  
 exit(0);  
 default:  
 printf("Invalid choice\n");  
 }  
 }  
 return 0;  
}

### Output:



## 37. Write a Program to Perform Pop, Push, Traverse operations on the stack using Pointer (Dynamic Stack).

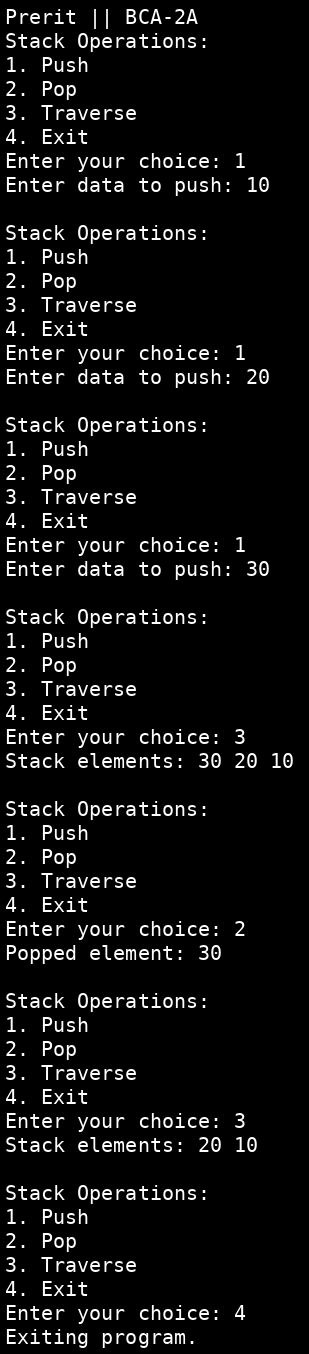
### Algorithm

```  
//Algorithm for Stack Operations (PUSH, POP, TRAVERSE)  
  
// GLOBAL: Node structure {data, next pointer}, TOP pointer  
  
//PUSH Algorithm (data)  
1. ALLOCATE new Node, SET newNode  
2. IF newNode == NULL THEN  
 PRINT "Stack Overflow"  
 EXIT  
 [END OF IF]  
3. SET newNode→data = data  
4. SET newNode→next = TOP  
5. SET TOP = newNode  
6. EXIT  
  
//POP Algorithm  
1. IF TOP == NULL THEN  
 PRINT "Stack Underflow"  
 EXIT  
 [END OF IF]  
2. SET temp = TOP  
3. SET TOP = TOP→next  
4. PRINT "Popped element: ", temp→data  
5. FREE temp  
6. EXIT  
  
//TRAVERSE Algorithm  
1. IF TOP == NULL THEN  
 PRINT "Stack is empty"  
 EXIT  
 [END OF IF]  
2. SET temp = TOP  
3. PRINT "Stack elements: "  
4. WHILE temp != NULL  
 PRINT temp→data, " "  
 SET temp = temp→next  
 [END OF LOOP]  
5. PRINT newline  
6. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node\* next;  
};  
  
struct Node\* top = NULL;  
  
void push(int data) {  
 struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  
 if (newNode == NULL) {  
 printf("Stack Overflow\n");  
 return;  
 }  
 newNode->data = data;  
 newNode->next = top;  
 top = newNode;  
}  
  
void pop() {  
 if (top == NULL) {  
 printf("Stack Underflow\n");  
 return;  
 }  
 struct Node\* temp = top;  
 top = top->next;  
 printf("Popped element: %d\n", temp->data);  
 free(temp);  
}  
  
void traverse() {  
 if (top == NULL) {  
 printf("Stack is empty\n");  
 return;  
 }  
 struct Node\* temp = top;  
 printf("Stack elements: ");  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, data;  
  
 do {  
 printf("\nStack Operations:\n");  
 printf("1. Push\n");  
 printf("2. Pop\n");  
 printf("3. Traverse\n");  
 printf("4. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to push: ");  
 scanf("%d", &data);  
 push(data);  
 break;  
 case 2:  
 pop();  
 break;  
 case 3:  
 traverse();  
 break;  
 case 4:  
 printf("Exiting program.\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 4);  
  
 return 0;  
}

### Output:



## 38. WAP to perform different operations such as Insert, Delete, Display elements using Dynamic Queue.

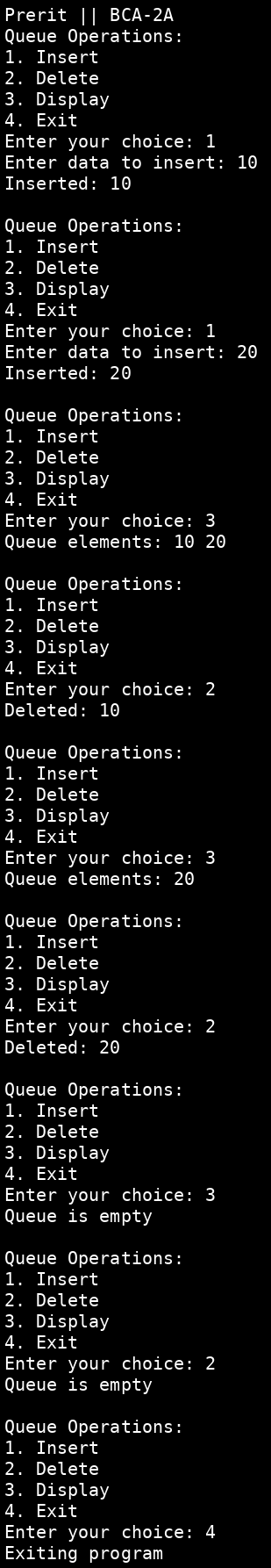
### Algorithm

```  
// Algorithm for Queue Operations using Linked List  
  
// Structure Definitions:  
// NODE: Structure with DATA (integer) and NEXT (pointer to NODE)  
// QUEUE: Structure with FRONT (pointer to NODE) and REAR (pointer to NODE)  
  
// Initialization  
1. INITIALIZE QUEUE.FRONT to NULL.  
2. INITIALIZE QUEUE.REAR to NULL.  
  
// Function: isEmpty(QUEUE)  
3. IF QUEUE.FRONT is NULL THEN  
 RETURN TRUE.  
 ELSE  
 RETURN FALSE.  
 [END OF IF]  
  
// Function: INSERT(QUEUE, DATA)  
4. ALLOCATE MEMORY for NEW\_NODE of type NODE.  
5. IF allocation FAILS THEN  
 PRINT "Memory allocation failed".  
 EXIT.  
 [END OF IF]  
6. SET NEW\_NODE→DATA to DATA.  
7. SET NEW\_NODE→NEXT to NULL.  
8. IF isEmpty(QUEUE) is TRUE THEN  
 SET QUEUE.FRONT to NEW\_NODE.  
 SET QUEUE.REAR to NEW\_NODE.  
 ELSE  
 SET QUEUE.REAR→NEXT to NEW\_NODE.  
 SET QUEUE.REAR to NEW\_NODE.  
 [END OF IF]  
9. PRINT "Inserted: DATA".  
  
// Function: DELETE(QUEUE)  
10. IF isEmpty(QUEUE) is TRUE THEN  
 PRINT "Queue is empty".  
 EXIT.  
 [END OF IF]  
11. SET TEMP to QUEUE.FRONT.  
12. SET QUEUE.FRONT to QUEUE.FRONT→NEXT.  
13. IF QUEUE.FRONT is NULL THEN  
 SET QUEUE.REAR to NULL.  
 [END OF IF]  
14. PRINT "Deleted: TEMP→DATA".  
15. FREE(TEMP).  
  
// Function: DISPLAY(QUEUE)  
16. IF isEmpty(QUEUE) is TRUE THEN  
 PRINT "Queue is empty".  
 EXIT.  
 [END OF IF]  
17. SET TEMP to QUEUE.FRONT.  
18. PRINT "Queue elements: ".  
19. WHILE TEMP is NOT NULL DO  
 PRINT TEMP→DATA.  
 SET TEMP to TEMP→NEXT.  
 [END OF LOOP]  
20. PRINT newline.  
  
// Main Function  
21. DO  
 PRINT "Queue Operations Menu".  
 READ CHOICE.  
 SWITCH (CHOICE)  
 CASE 1:  
 READ DATA.  
 CALL INSERT(QUEUE, DATA).  
 BREAK.  
 CASE 2:  
 CALL DELETE(QUEUE).  
 BREAK.  
 CASE 3:  
 CALL DISPLAY(QUEUE).  
 BREAK.  
 CASE 4:  
 PRINT "Exiting program".  
 BREAK.  
 DEFAULT:  
 PRINT "Invalid choice".  
 END SWITCH  
 WHILE CHOICE is NOT 4.  
22. EXIT.  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node\* next;  
};  
  
struct Queue {  
 struct Node\* front;  
 struct Node\* rear;  
};  
  
int isEmpty(struct Queue\* queue) {  
 return (queue->front == NULL);  
}  
  
void insert(struct Queue\* queue, int data) {  
 struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  
 if (newNode == NULL) {  
 printf("Memory allocation failed\n");  
 return;  
 }  
 newNode->data = data;  
 newNode->next = NULL;  
  
 if (isEmpty(queue)) {  
 queue->front = newNode;  
 queue->rear = newNode;  
 } else {  
 queue->rear->next = newNode;  
 queue->rear = newNode;  
 }  
 printf("Inserted: %d\n", data);  
}  
  
void delete(struct Queue\* queue) {  
 if (isEmpty(queue)) {  
 printf("Queue is empty\n");  
 return;  
 }  
  
 struct Node\* temp = queue->front;  
 queue->front = queue->front->next;  
  
 if (queue->front == NULL) {  
 queue->rear = NULL;  
 }  
  
 printf("Deleted: %d\n", temp->data);  
 free(temp);  
}  
  
void display(struct Queue\* queue) {  
 if (isEmpty(queue)) {  
 printf("Queue is empty\n");  
 return;  
 }  
  
 struct Node\* temp = queue->front;  
 printf("Queue elements: ");  
 while (temp != NULL) {  
 printf("%d ", temp->data);  
 temp = temp->next;  
 }  
 printf("\n");  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 struct Queue queue;  
 queue.front = NULL;  
 queue.rear = NULL;  
 int choice, data;  
  
 do {  
 printf("\nQueue Operations:\n");  
 printf("1. Insert\n");  
 printf("2. Delete\n");  
 printf("3. Display\n");  
 printf("4. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 insert(&queue, data);  
 break;  
 case 2:  
 delete(&queue);  
 break;  
 case 3:  
 display(&queue);  
 break;  
 case 4:  
 printf("Exiting program\n");  
 break;  
 default:  
 printf("Invalid choice\n");  
 }  
 } while (choice != 4);  
  
 return 0;  
}

### Output:



## 39. WAP to implement priority queue with three priority values (1: Lowest, 2, 3:Highest)

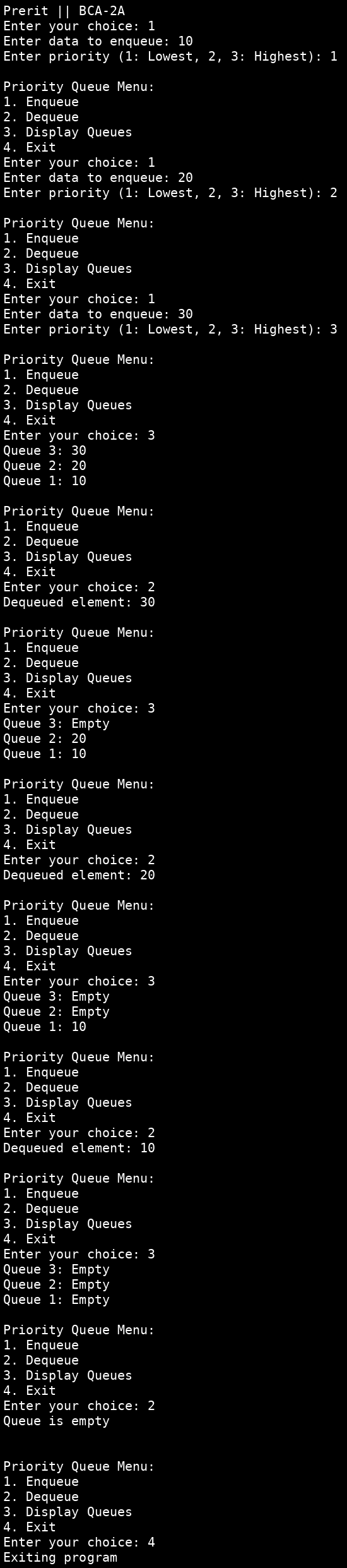
### Algorithm

```  
// Algorithm for Priority Queue using Multiple Arrays  
  
// ENQUEUE(data, priority)  
1. IF priority IS 1 THEN  
2. IF rear1 IS 99 THEN  
3. PRINT "Queue 1 is full"  
4. ELSE  
5. IF front1 IS -1 THEN  
6. SET front1 = 0  
7. [END OF IF]  
8. SET rear1 = rear1 + 1  
9. SET queue1[rear1] = data  
10. [END OF IF]  
11. ELSE IF priority IS 2 THEN  
12. IF rear2 IS 99 THEN  
13. PRINT "Queue 2 is full"  
14. ELSE  
15. IF front2 IS -1 THEN  
16. SET front2 = 0  
17. [END OF IF]  
18. SET rear2 = rear2 + 1  
19. SET queue2[rear2] = data  
20. [END OF IF]  
21. ELSE IF priority IS 3 THEN  
22. IF rear3 IS 99 THEN  
23. PRINT "Queue 3 is full"  
24. ELSE  
25. IF front3 IS -1 THEN  
26. SET front3 = 0  
27. [END OF IF]  
28. SET rear3 = rear3 + 1  
29. SET queue3[rear3] = data  
30. [END OF IF]  
31. ELSE  
32. PRINT "Invalid priority"  
33. [END OF IF]  
34. EXIT  
  
// DEQUEUE()  
1. IF front3 IS NOT -1 THEN  
2. SET data = queue3[front3]  
3. SET front3 = front3 + 1  
4. IF front3 > rear3 THEN  
5. SET front3 = -1  
6. SET rear3 = -1  
7. [END OF IF]  
8. RETURN data  
9. ELSE IF front2 IS NOT -1 THEN  
10. SET data = queue2[front2]  
11. SET front2 = front2 + 1  
12. IF front2 > rear2 THEN  
13. SET front2 = -1  
14. SET rear2 = -1  
15. [END OF IF]  
16. RETURN data  
17. ELSE IF front1 IS NOT -1 THEN  
18. SET data = queue1[front1]  
19. SET front1 = front1 + 1  
20. IF front1 > rear1 THEN  
21. SET front1 = -1  
22. SET rear1 = -1  
23. [END OF IF]  
24. RETURN data  
25. ELSE  
26. PRINT "Queue is empty"  
27. RETURN -1  
28. [END OF IF]  
29. EXIT  
  
// DISPLAYQUEUES()  
1. PRINT "Queue 3: "  
2. IF front3 IS -1 THEN  
3. PRINT "Empty"  
4. ELSE  
5. FOR i = front3 TO rear3  
6. PRINT queue3[i]  
7. [END OF LOOP]  
8. [END OF IF]  
9. PRINT "Queue 2: "  
10.IF front2 IS -1 THEN  
11. PRINT "Empty"  
12.ELSE  
13. FOR i = front2 TO rear2  
14. PRINT queue2[i]  
15. [END OF LOOP]  
16.[END OF IF]  
17.PRINT "Queue 1: "  
18.IF front1 IS -1 THEN  
19. PRINT "Empty"  
20.ELSE  
21. FOR i = front1 TO rear1  
22. PRINT queue1[i]  
23. [END OF LOOP]  
24.[END OF IF]  
25.EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 int priority;  
};  
  
int queue1[100], queue2[100], queue3[100];  
int front1 = -1, rear1 = -1;  
int front2 = -1, rear2 = -1;  
int front3 = -1, rear3 = -1;  
  
void enqueue(int data, int priority) {  
 if (priority == 1) {  
 if (rear1 == 99) printf("Queue 1 is full\n");  
 else {  
 if (front1 == -1) front1 = 0;  
 queue1[++rear1] = data;  
 }  
 } else if (priority == 2) {  
 if (rear2 == 99) printf("Queue 2 is full\n");  
 else {  
 if (front2 == -1) front2 = 0;  
 queue2[++rear2] = data;  
 }  
 } else if (priority == 3) {  
 if (rear3 == 99) printf("Queue 3 is full\n");  
 else {  
 if (front3 == -1) front3 = 0;  
 queue3[++rear3] = data;  
 }  
 } else {  
 printf("Invalid priority\n");  
 }  
}  
  
int dequeue() {  
 if (front3 != -1) {  
 int data = queue3[front3++];  
 if (front3 > rear3) front3 = rear3 = -1;  
 return data;  
 } else if (front2 != -1) {  
 int data = queue2[front2++];  
 if (front2 > rear2) front2 = rear2 = -1;  
 return data;  
 } else if (front1 != -1) {  
 int data = queue1[front1++];  
 if (front1 > rear1) front1 = rear1 = -1;  
 return data;  
 } else {  
 printf("Queue is empty\n");  
 return -1;  
 }  
}  
  
void displayQueues() {  
 printf("Queue 3: ");  
 if (front3 == -1) printf("Empty\n");  
 else {  
 for (int i = front3; i <= rear3; i++) {  
 printf("%d ", queue3[i]);  
 }  
 printf("\n");  
 }  
 printf("Queue 2: ");  
 if (front2 == -1) printf("Empty\n");  
 else {  
 for (int i = front2; i <= rear2; i++) {  
 printf("%d ", queue2[i]);  
 }  
 printf("\n");  
 }  
 printf("Queue 1: ");  
 if (front1 == -1) printf("Empty\n");  
 else {  
 for (int i = front1; i <= rear1; i++) {  
 printf("%d ", queue1[i]);  
 }  
 printf("\n");  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int choice, data, priority;  
  
 do {  
 printf("\nPriority Queue Menu:\n");  
 printf("1. Enqueue\n");  
 printf("2. Dequeue\n");  
 printf("3. Display Queues\n");  
 printf("4. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to enqueue: ");  
 scanf("%d", &data);  
 printf("Enter priority (1: Lowest, 2, 3: Highest): ");  
 scanf("%d", &priority);  
 enqueue(data, priority);  
 break;  
 case 2:  
 data = dequeue();  
 if (data != -1) {  
 printf("Dequeued element: %d\n", data);  
 }  
 break;  
 case 3:  
 displayQueues();  
 break;  
 case 4:  
 printf("Exiting program\n");  
 break;  
 default:  
 printf("Invalid choice\n");  
 }  
 } while (choice != 4);  
  
 return 0;  
}

### Output:



## 40. WAP to find factorial of a number using recursion

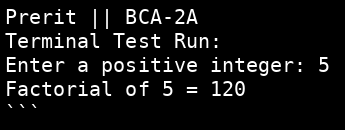
### Algorithm

\*\*Algorithm for Factorial Calculation\*\*  
  
\*\*1. INITIALIZE:\*\* START.  
\*\*2. OUTPUT:\*\* "Prerit || BCA-2A"  
\*\*3. OUTPUT:\*\* "Enter a positive integer: "  
\*\*4. INPUT:\*\* num  
\*\*5. IF:\*\* num < 0 THEN  
 \* OUTPUT: "Factorial is not defined for negative numbers."  
 [END OF IF]  
\*\*6. ELSE:\*\*  
 \* CALL: Factorial(num) AND STORE the returned value into result.  
 \* OUTPUT: "Factorial of num = result"  
 [END OF ELSE]  
\*\*7. EXIT.\*\*  
  
\*\*Algorithm for Factorial (Recursive Function)\*\*  
  
\*\*1. FUNCTION:\*\* Factorial(n)  
\*\*2. IF:\*\* n == 0 THEN  
 \* RETURN: 1  
 [END OF IF]  
\*\*3. ELSE:\*\*  
 \* RETURN: n \* Factorial(n - 1)  
 [END OF ELSE]  
\*\*4. END FUNCTION.\*\*

### Code:

#include <stdio.h>  
  
int factorial(int n) {  
 if (n == 0) {  
 return 1;  
 } else {  
 return n \* factorial(n - 1);  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int num;  
 printf("\nEnter a positive integer: ");  
 scanf("%d", &num);  
 if (num < 0) {  
 printf("Factorial is not defined for negative numbers.\n");  
 } else {  
 printf("Factorial of %d = %d\n", num, factorial(num));  
 }  
 return 0;  
}

### Output:



## 41. WAP to generate Fibonacci Series of n-terms

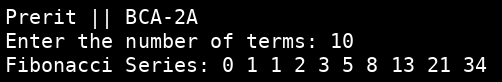
### Algorithm

\*\*Algorithm: Fibonacci Series Generation\*\*  
  
Step 1: PRINT "Prerit || BCA-2A"  
Step 2: SET A = 0  
Step 3: SET B = 1  
Step 4: INPUT N (number of terms)  
Step 5: PRINT "Fibonacci Series: "  
Step 6: PRINT A, PRINT B  
Step 7: FOR I = 3 TO N  
 Step 8: SET NEXTTERM = A + B  
 Step 9: PRINT NEXTTERM  
 Step 10: SET A = B  
 Step 11: SET B = NEXTTERM  
 [END OF LOOP]  
Step 12: EXIT

### Code:

#include <stdio.h>  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int n, i, a = 0, b = 1, nextTerm;  
  
 printf("Enter the number of terms: ");  
 scanf("%d", &n);  
  
 printf("Fibonacci Series: ");  
 printf("%d %d ", a, b);  
  
 for (i = 3; i <= n; ++i) {  
 nextTerm = a + b;  
 printf("%d ", nextTerm);  
 a = b;  
 b = nextTerm;  
 }  
  
 return 0;  
}

### Output:



## 42. Write a program to calculate the GCD of two numbers using recursive functions

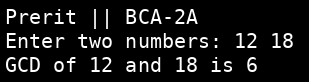
### Algorithm

\*\*Algorithm for Calculating GCD\*\*  
  
\*\*GCD Function (Recursive)\*\*  
  
1. FUNCTION GCD(a, b)  
2. IF b = 0 THEN  
 \* RETURN a  
 [END OF IF]  
3. ELSE  
 \* RETURN GCD(b, a MOD b)  
 [END OF ELSE]  
4. END FUNCTION  
  
\*\*Main Function\*\*  
  
1. FUNCTION MAIN  
2. PRINT "Prerit || BCA-2A"  
3. DECLARE num1, num2 AS INTEGER  
4. PRINT "Enter two numbers: "  
5. READ num1, num2  
6. PRINT "GCD of ", num1, " and ", num2, " is ", GCD(num1, num2)  
7. EXIT  
8. END FUNCTION

### Code:

#include <stdio.h>  
  
int gcd(int a, int b) {  
 if (b == 0) {  
 return a;  
 } else {  
 return gcd(b, a % b);  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 int num1, num2;  
 printf("Enter two numbers: ");  
 scanf("%d %d", &num1, &num2);  
 printf("GCD of %d and %d is %d\n", num1, num2, gcd(num1, num2));  
 return 0;  
}

### Output:



## 43. WAP to create a Binary tree and traverse the tree in Inorder, Preorder and Postorder manner using recursive functions

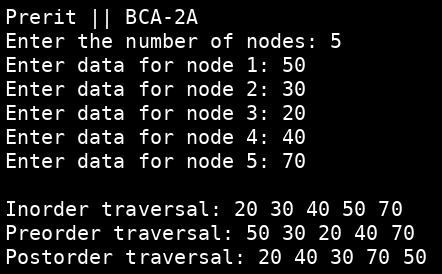
### Algorithm

```  
// Algorithm for Binary Search Tree Operations  
  
// Function: CREATE\_NODE(data)  
// Creates a new node with the given data.  
1. ALLOCATE memory for NEW\_NODE of type Node.  
2. SET NEW\_NODE→DATA = data.  
3. SET NEW\_NODE→LEFT = NULL.  
4. SET NEW\_NODE→RIGHT = NULL.  
5. RETURN NEW\_NODE.  
  
// Function: INSERT\_NODE(root, data)  
// Inserts a new node with the given data into the BST rooted at 'root'.  
1. IF root is NULL THEN  
 2. RETURN CREATE\_NODE(data).  
 [END OF IF]  
3. IF data < root→DATA THEN  
 4. SET root→LEFT = INSERT\_NODE(root→LEFT, data).  
 [END OF IF]  
4. ELSE IF data > root→DATA THEN  
 5. SET root→RIGHT = INSERT\_NODE(root→RIGHT, data).  
 [END OF IF]  
5. RETURN root.  
  
// Function: INORDER\_TRAVERSAL(root)  
// Performs an inorder traversal of the BST rooted at 'root'.  
1. IF root is not NULL THEN  
 2. INORDER\_TRAVERSAL(root→LEFT).  
 3. PRINT root→DATA.  
 4. INORDER\_TRAVERSAL(root→RIGHT).  
 [END OF IF]  
  
// Function: PREORDER\_TRAVERSAL(root)  
// Performs a preorder traversal of the BST rooted at 'root'.  
1. IF root is not NULL THEN  
 2. PRINT root→DATA.  
 3. PREORDER\_TRAVERSAL(root→LEFT).  
 4. PREORDER\_TRAVERSAL(root→RIGHT).  
 [END OF IF]  
  
// Function: POSTORDER\_TRAVERSAL(root)  
// Performs a postorder traversal of the BST rooted at 'root'.  
1. IF root is not NULL THEN  
 2. POSTORDER\_TRAVERSAL(root→LEFT).  
 3. POSTORDER\_TRAVERSAL(root→RIGHT).  
 4. PRINT root→DATA.  
 [END OF IF]  
  
// Function: MAIN  
1. SET root = NULL.  
2. INPUT n (number of nodes).  
3. FOR i = 0 to n-1  
 4. INPUT data for node i+1.  
 5. SET root = INSERT\_NODE(root, data).  
 [END OF LOOP]  
4. PRINT "Inorder traversal:".  
5. INORDER\_TRAVERSAL(root).  
6. PRINT "Preorder traversal:".  
7. PREORDER\_TRAVERSAL(root).  
8. PRINT "Postorder traversal:".  
9. POSTORDER\_TRAVERSAL(root).  
10. EXIT.  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*left;  
 struct Node \*right;  
};  
  
struct Node\* createNode(int data) {  
 struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->left = NULL;  
 newNode->right = NULL;  
 return newNode;  
}  
  
struct Node\* insertNode(struct Node\* root, int data) {  
 if (root == NULL) {  
 return createNode(data);  
 }  
 if (data < root->data) {  
 root->left = insertNode(root->left, data);  
 } else if (data > root->data) {  
 root->right = insertNode(root->right, data);  
 }  
 return root;  
}  
  
void inorderTraversal(struct Node\* root) {  
 if (root != NULL) {  
 inorderTraversal(root->left);  
 printf("%d ", root->data);  
 inorderTraversal(root->right);  
 }  
}  
  
void preorderTraversal(struct Node\* root) {  
 if (root != NULL) {  
 printf("%d ", root->data);  
 preorderTraversal(root->left);  
 preorderTraversal(root->right);  
 }  
}  
  
void postorderTraversal(struct Node\* root) {  
 if (root != NULL) {  
 postorderTraversal(root->left);  
 postorderTraversal(root->right);  
 printf("%d ", root->data);  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 struct Node\* root = NULL;  
 int n, data, i;  
  
 printf("\nEnter the number of nodes: ");  
 scanf("%d", &n);  
  
 for (i = 0; i < n; i++) {  
 printf("Enter data for node %d: ", i + 1);  
 scanf("%d", &data);  
 root = insertNode(root, data);  
 }  
  
 printf("\nInorder traversal: ");  
 inorderTraversal(root);  
 printf("\n");  
  
 printf("Preorder traversal: ");  
 preorderTraversal(root);  
 printf("\n");  
  
 printf("Postorder traversal: ");  
 postorderTraversal(root);  
 printf("\n");  
  
 return 0;  
}

### Output:



## 44. WAP to create a Binary Search Tree (BST) and traverse the tree in Inorder, Preorder and Postorder manner using recursive functions

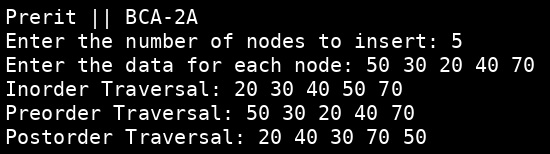
### Algorithm

```  
// Algorithm for Binary Search Tree Operations  
  
// 1. START  
  
// 2. PROCEDURE CreateNode(data)  
// 2.1. ALLOCATE memory for NEW\_NODE  
// 2.2. NEW\_NODE→data = data  
// 2.3. NEW\_NODE→left = NULL  
// 2.4. NEW\_NODE→right = NULL  
// 2.5. RETURN NEW\_NODE  
  
// 3. PROCEDURE InsertNode(ROOT, data)  
// 3.1. IF ROOT = NULL THEN  
// 3.1.1. RETURN CreateNode(data)  
// [END OF IF]  
// 3.2. IF data < ROOT→data THEN  
// 3.2.1. ROOT→left = InsertNode(ROOT→left, data)  
// ELSE IF data > ROOT→data THEN  
// 3.2.2. ROOT→right = InsertNode(ROOT→right, data)  
// [END OF IF]  
// 3.3. RETURN ROOT  
  
// 4. PROCEDURE InorderTraversal(ROOT)  
// 4.1. IF ROOT != NULL THEN  
// 4.1.1. InorderTraversal(ROOT→left)  
// 4.1.2. PRINT ROOT→data  
// 4.1.3. InorderTraversal(ROOT→right)  
// [END OF IF]  
  
// 5. PROCEDURE PreorderTraversal(ROOT)  
// 5.1. IF ROOT != NULL THEN  
// 5.1.1. PRINT ROOT→data  
// 5.1.2. PreorderTraversal(ROOT→left)  
// 5.1.3. PreorderTraversal(ROOT→right)  
// [END OF IF]  
  
// 6. PROCEDURE PostorderTraversal(ROOT)  
// 6.1. IF ROOT != NULL THEN  
// 6.1.1. PostorderTraversal(ROOT→left)  
// 6.1.2. PostorderTraversal(ROOT→right)  
// 6.1.3. PRINT ROOT→data  
// [END OF IF]  
  
// 7. PROCEDURE Main()  
// 7.1. SET ROOT = NULL  
// 7.2. INPUT number of nodes, N  
// 7.3. FOR I = 0 to N - 1 DO  
// 7.3.1. INPUT data  
// 7.3.2. ROOT = InsertNode(ROOT, data)  
// [END OF LOOP]  
// 7.4. PRINT "Inorder Traversal:"  
// 7.5. InorderTraversal(ROOT)  
// 7.6. PRINT "Preorder Traversal:"  
// 7.7. PreorderTraversal(ROOT)  
// 7.8. PRINT "Postorder Traversal:"  
// 7.9. PostorderTraversal(ROOT)  
  
// 8. EXIT  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
  
struct Node {  
 int data;  
 struct Node \*left;  
 struct Node \*right;  
};  
  
struct Node\* createNode(int data) {  
 struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->left = NULL;  
 newNode->right = NULL;  
 return newNode;  
}  
  
struct Node\* insertNode(struct Node \*root, int data) {  
 if (root == NULL) {  
 return createNode(data);  
 }  
 if (data < root->data) {  
 root->left = insertNode(root->left, data);  
 } else if (data > root->data) {  
 root->right = insertNode(root->right, data);  
 }  
 return root;  
}  
  
void inorderTraversal(struct Node \*root) {  
 if (root != NULL) {  
 inorderTraversal(root->left);  
 printf("%d ", root->data);  
 inorderTraversal(root->right);  
 }  
}  
  
void preorderTraversal(struct Node \*root) {  
 if (root != NULL) {  
 printf("%d ", root->data);  
 preorderTraversal(root->left);  
 preorderTraversal(root->right);  
 }  
}  
  
void postorderTraversal(struct Node \*root) {  
 if (root != NULL) {  
 postorderTraversal(root->left);  
 postorderTraversal(root->right);  
 printf("%d ", root->data);  
 }  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 struct Node \*root = NULL;  
 int n, data, i;  
  
 printf("\nEnter the number of nodes to insert: ");  
 scanf("%d", &n);  
  
 printf("Enter the data for each node: ");  
 for (i = 0; i < n; i++) {  
 scanf("%d", &data);  
 root = insertNode(root, data);  
 }  
  
 printf("Inorder Traversal: ");  
 inorderTraversal(root);  
 printf("\n");  
  
 printf("Preorder Traversal: ");  
 preorderTraversal(root);  
 printf("\n");  
  
 printf("Postorder Traversal: ");  
 postorderTraversal(root);  
 printf("\n");  
  
 return 0;  
}

### Output:



## 45. WAP to implement following recursive operations on a Binary Search Tree (BST) a. Find an element b. Insert an element c. Delete an element d. Count the number e. Find maximum element f. Find minimum element g. Find height of the tree

### Algorithm

```  
// Algorithm for Binary Search Tree Operations  
  
// ---- INSERT Operation ----  
1. IF ROOT is NULL, THEN  
 2. CREATE a new NODE with DATA.  
 3. RETURN the new NODE.  
 [END OF IF]  
4. IF DATA < ROOT→DATA, THEN  
 5. ROOT→LEFT = INSERT(ROOT→LEFT, DATA).  
 [END OF IF]  
5. ELSE IF DATA > ROOT→DATA, THEN  
 6. ROOT→RIGHT = INSERT(ROOT→RIGHT, DATA).  
 [END OF IF]  
6. RETURN ROOT.  
  
// ---- FINDMIN Operation ----  
1. SET CURRENT = NODE.  
2. WHILE CURRENT is not NULL AND CURRENT→LEFT is not NULL, DO  
 3. SET CURRENT = CURRENT→LEFT.  
 [END OF LOOP]  
3. RETURN CURRENT.  
  
// ---- DELETE Operation ----  
1. IF ROOT is NULL, THEN  
 2. RETURN ROOT.  
 [END OF IF]  
2. IF DATA < ROOT→DATA, THEN  
 3. ROOT→LEFT = DELETE(ROOT→LEFT, DATA).  
 [END OF IF]  
3. ELSE IF DATA > ROOT→DATA, THEN  
 4. ROOT→RIGHT = DELETE(ROOT→RIGHT, DATA).  
 [END OF IF]  
4. ELSE  
 5. IF ROOT→LEFT is NULL, THEN  
 6. SET TEMP = ROOT→RIGHT.  
 7. FREE ROOT.  
 8. RETURN TEMP.  
 [END OF IF]  
 6. ELSE IF ROOT→RIGHT is NULL, THEN  
 7. SET TEMP = ROOT→LEFT.  
 8. FREE ROOT.  
 9. RETURN TEMP.  
 [END OF IF]  
 7. SET TEMP = FINDMIN(ROOT→RIGHT).  
 8. SET ROOT→DATA = TEMP→DATA.  
 9. ROOT→RIGHT = DELETE(ROOT→RIGHT, TEMP→DATA).  
 [END OF ELSE]  
5. RETURN ROOT.  
  
// ---- SEARCH Operation ----  
1. IF ROOT is NULL, THEN  
 2. RETURN 0.  
 [END OF IF]  
2. IF ROOT→DATA = DATA, THEN  
 3. RETURN 1.  
 [END OF IF]  
3. ELSE IF DATA < ROOT→DATA, THEN  
 4. RETURN SEARCH(ROOT→LEFT, DATA).  
 [END OF IF]  
4. ELSE  
 5. RETURN SEARCH(ROOT→RIGHT, DATA).  
 [END OF IF]  
  
// ---- COUNTNODES Operation ----  
1. IF ROOT is NULL, THEN  
 2. RETURN 0.  
 [END OF IF]  
2. RETURN 1 + COUNTNODES(ROOT→LEFT) + COUNTNODES(ROOT→RIGHT).  
  
// ---- FINDMAX Operation ----  
1. IF ROOT is NULL, THEN  
 2. RETURN INT\_MIN.  
 [END OF IF]  
2. SET CURRENT = ROOT.  
3. WHILE CURRENT→RIGHT is not NULL, DO  
 4. SET CURRENT = CURRENT→RIGHT.  
 [END OF LOOP]  
4. RETURN CURRENT→DATA.  
  
// ---- FINDMINVAL Operation ----  
1. IF ROOT is NULL, THEN  
 2. RETURN INT\_MAX.  
 [END OF IF]  
2. SET CURRENT = ROOT.  
3. WHILE CURRENT→LEFT is not NULL, DO  
 4. SET CURRENT = CURRENT→LEFT.  
 [END OF LOOP]  
4. RETURN CURRENT→DATA.  
  
// ---- FINDHEIGHT Operation ----  
1. IF ROOT is NULL, THEN  
 2. RETURN -1.  
 [END OF IF]  
2. SET LEFTHEIGHT = FINDHEIGHT(ROOT→LEFT).  
3. SET RIGHTHEIGHT = FINDHEIGHT(ROOT→RIGHT).  
4. IF LEFTHEIGHT > RIGHTHEIGHT, THEN  
 5. RETURN 1 + LEFTHEIGHT.  
 [END OF IF]  
5. ELSE  
 6. RETURN 1 + RIGHTHEIGHT.  
 [END OF ELSE]  
```

### Code:

#include <stdio.h>  
#include <stdlib.h>  
#include <limits.h>  
  
struct Node {  
 int data;  
 struct Node \*left;  
 struct Node \*right;  
};  
  
struct Node\* createNode(int data) {  
 struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  
 newNode->data = data;  
 newNode->left = NULL;  
 newNode->right = NULL;  
 return newNode;  
}  
  
struct Node\* insert(struct Node\* root, int data) {  
 if (root == NULL) {  
 return createNode(data);  
 }  
 if (data < root->data) {  
 root->left = insert(root->left, data);  
 } else if (data > root->data) {  
 root->right = insert(root->right, data);  
 }  
 return root;  
}  
  
struct Node\* findMin(struct Node\* node) {  
 struct Node\* current = node;  
 while (current && current->left != NULL) {  
 current = current->left;  
 }  
 return current;  
}  
  
struct Node\* deleteNode(struct Node\* root, int data) {  
 if (root == NULL) {  
 return root;  
 }  
 if (data < root->data) {  
 root->left = deleteNode(root->left, data);  
 } else if (data > root->data) {  
 root->right = deleteNode(root->right, data);  
 } else {  
 if (root->left == NULL) {  
 struct Node\* temp = root->right;  
 free(root);  
 return temp;  
 } else if (root->right == NULL) {  
 struct Node\* temp = root->left;  
 free(root);  
 return temp;  
 }  
 struct Node\* temp = findMin(root->right);  
 root->data = temp->data;  
 root->right = deleteNode(root->right, temp->data);  
 }  
 return root;  
}  
  
int search(struct Node\* root, int data) {  
 if (root == NULL) {  
 return 0;  
 }  
 if (root->data == data) {  
 return 1;  
 } else if (data < root->data) {  
 return search(root->left, data);  
 } else {  
 return search(root->right, data);  
 }  
}  
  
int countNodes(struct Node\* root) {  
 if (root == NULL) {  
 return 0;  
 }  
 return 1 + countNodes(root->left) + countNodes(root->right);  
}  
  
int findMax(struct Node\* root) {  
 if (root == NULL) {  
 return INT\_MIN;  
 }  
 struct Node\* current = root;  
 while (current->right != NULL) {  
 current = current->right;  
 }  
 return current->data;  
}  
  
int findMinVal(struct Node\* root) {  
 if (root == NULL) {  
 return INT\_MAX;  
 }  
 struct Node\* current = root;  
 while (current->left != NULL) {  
 current = current->left;  
 }  
 return current->data;  
}  
  
int findHeight(struct Node\* root) {  
 if (root == NULL) {  
 return -1;  
 }  
 int leftHeight = findHeight(root->left);  
 int rightHeight = findHeight(root->right);  
 return 1 + (leftHeight > rightHeight ? leftHeight : rightHeight);  
}  
  
int main() {  
 printf("Prerit || BCA-2A");  
 struct Node\* root = NULL;  
 int choice, data;  
  
 do {  
 printf("\nBinary Search Tree Operations:\n");  
 printf("1. Insert\n");  
 printf("2. Delete\n");  
 printf("3. Search\n");  
 printf("4. Count Nodes\n");  
 printf("5. Find Maximum\n");  
 printf("6. Find Minimum\n");  
 printf("7. Find Height\n");  
 printf("0. Exit\n");  
 printf("Enter your choice: ");  
 scanf("%d", &choice);  
  
 switch (choice) {  
 case 1:  
 printf("Enter data to insert: ");  
 scanf("%d", &data);  
 root = insert(root, data);  
 break;  
 case 2:  
 printf("Enter data to delete: ");  
 scanf("%d", &data);  
 root = deleteNode(root, data);  
 break;  
 case 3:  
 printf("Enter data to search: ");  
 scanf("%d", &data);  
 if (search(root, data)) {  
 printf("Element %d found in the BST.\n", data);  
 } else {  
 printf("Element %d not found in the BST.\n", data);  
 }  
 break;  
 case 4:  
 printf("Number of nodes in the BST: %d\n", countNodes(root));  
 break;  
 case 5:  
 data = findMax(root);  
 if (data == INT\_MIN) {  
 printf("Tree is empty\n");  
 } else {  
 printf("Maximum element in the BST: %d\n", data);  
 }  
 break;  
 case 6:  
 data = findMinVal(root);  
 if (data == INT\_MAX) {  
 printf("Tree is empty\n");  
 } else {  
 printf("Minimum element in the BST: %d\n", data);  
 }  
 break;  
 case 7:  
 printf("Height of the BST: %d\n", findHeight(root));  
 break;  
 case 0:  
 printf("Exiting...\n");  
 break;  
 default:  
 printf("Invalid choice. Please try again.\n");  
 }  
 } while (choice != 0);  
  
 return 0;  
}

### Output:

