

1. ARRAY AND SUM AND AVERAGE

Programiz C Online Compiler

main.c



Run

Output

```
1 #include <stdio.h>
2 int main() {
3     int arr[100], n, i, sum = 0;
4     float avg;
5     printf("Enter number of elements: ");
6     scanf("%d", &n);
7     printf("Enter %d elements:\n", n);
8     for (i = 0; i < n; i++){
9         scanf("%d", &arr[i]);
10        sum = sum + arr[i];
11    }
12    avg = (float)sum / n;
13    printf("Sum = %d\nAverage = %.2f\n", sum, avg);
14    return 0;
15 }
```

```
Enter number of elements: 3
Enter 3 elements:
12
3
45
Sum = 60
Average = 20.00
```

=== Code Execution Successful ===

2. TRAVERSING IN ARRAY

main.c



Run

Output

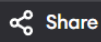
```
1 #include <stdio.h>
2 int main() {
3     int arr[10], n;
4     printf("Enter the value of elements:");
5     scanf("%d", &n);
6     printf("Enter %d elements:", n);
7     for (int i = 0; i < n; i++){
8         scanf("%d", &arr[i]);
9     }
10    printf("Traversing a elements:");
11    for (int i = 0; i < n; i++){
12        printf("%d ", arr[i]);
13    }
14    printf("\n");
15    return 0;
16 }
```

```
Enter the value of elements:4
Enter 4 elements:1
2
33
4
Traversing a elements:1 2 33 4
```

=== Code Execution Successful ===

3. INSERTION IN ARRAY

main.c



Run

Output

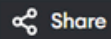
```
1 #include <stdio.h>
2 int main() {
3     int arr[10], n, value, k;
4     printf("ENTER THE VALUE OF ELEMENT: ");
5     scanf("%d", &n);
6     printf("ENTER %d elements:\n", n);
7     for (int i = 0; i < n; i++){
8         scanf("%d", &arr[i]);
9     }
10    printf("ENTER THE VALUE TO INSERT: ");
11    scanf("%d", &value);
12    printf("ENTER THE POSITION (0 to %d): ", n);
13    scanf("%d", &k);
14    for (int i = n; i > k; i--){
15        arr[i] = arr[i-1];
16    }
17    arr[k] = value;
18    printf("new array: ");
19    for (int i = 0; i <= n; i++){
20        printf("%d", arr[i]);
21    }
22    return 0;
23 }
24
```

```
ENTER THE VALUE OF ELEMENT: 5
ENTER 5 elements:
1
2
3
4
5
ENTER THE VALUE TO INSERT: 200
ENTER THE POSITION (0 to 5): 3
new array: 12320045

=== Code Execution Successful ===
```

4.DELETION IN ARRAY

main.c



Run

Output

```
1 #include <stdio.h>
2 int main() {
3     int arr[10], N, VALUE, i;
4     printf("Enter the number of elements: ");
5     scanf("%d", &N);
6     printf("Enter %d elements:\n", N);
7     for (i = 0; i < N; i++){
8         scanf("%d", &arr[i]);
9     }
10    printf("VALUE to delete (0 to %d): ", N - 1);
11    scanf("%d", &VALUE);
12    for (i = VALUE; i < N - 1; i++) {
13        arr[i] = arr[i + 1];
14    }
15    printf("NEW ARRAY: ");
16    for (i = 0; i < N - 1; i++){
17        printf("%d ", arr[i]);
18    }
19    return 0;
20 }
21
```




```
Enter the number of elements: 5
Enter 5 elements:
1
2
3
4
5
VALUE to delete (0 to 4): 3
NEW ARRAY: 1 2 3 5

=== Code Execution Successful ===
```

5.DECIMAL TO BINARY

Programiz C Online Compiler

main.c

 Share

Run

```
1 #include <stdio.h>
2 int main() {
3     int binary[100];
4     int num, index = 0;
5     printf("Enter a decimal number: ");
6     scanf("%d", &num);
7     while (num > 0){
8         binary[index] = num % 2;
9         num = num / 2;
10        index++;
11    }
12    printf("Binary: ");
13    for (int j = index - 1; j >= 0; j--){
14        printf("%d", binary[j]);
15    }
16    printf("\n");
17    return 0;
18 }
19
```

Output

Enter a decimal number: 9
Binary: 1001

=== Code Execution Successful ===

6.MULTIPLICATION IN ARRAY 2D

| main.c | Run | Output |
|--|-----|---|
| <pre> 1 #include <stdio.h> 2 int main() { 3 int A[2][2] = { {2, 3}, {6, 7} }; 4 int B[2][2] = { {6, 7}, {8, 9} }; 5 int result[2][2]; 6 for (int i = 0; i < 2; i++) { 7 for (int j = 0; j < 2; j++) { 8 result[i][j] = 0; 9 } 10 } 11 for (int i = 0; i < 2; i++) { 12 for (int j = 0; j < 2; j++) { 13 for (int k = 0; k < 2; k++) { 14 result[i][j] += A[i][k] * B[k][j]; 15 } 16 } 17 } 18 printf("Matrix result:\n"); 19 for (int i = 0; i < 2; i++) { 20 for (int j = 0; j < 2; j++) { 21 printf("%d ", result[i][j]); 22 } 23 printf("\n"); 24 } 25 return 0; </pre> | Run | <pre> Matrix result: 36 41 92 105 === Code Execution Successful === </pre> |

7.TRANSPOSING IN ARRAY

| main.c | Run | Output |
|--|-----|--|
| <pre> 1 #include <stdio.h> 2 int main() { 3 int matrix[10][10], transpose[10][10]; 4 int row, col; 5 printf("Enter the number of rows and columns: "); 6 scanf("%d %d", &row, &col); 7 printf("Enter elements of the matrix:\n"); 8 for (int i = 0; i < row; i++) { 9 for (int j = 0; j < col; j++) { 10 scanf("%d", &matrix[i][j]); 11 } 12 } 13 for (int i = 0; i < row; i++) { 14 for (int j = 0; j < col; j++) { 15 transpose[j][i] = matrix[i][j]; 16 } 17 } 18 printf("Transpose of the matrix:\n"); 19 for (int i = 0; i < col; i++) { 20 for (int j = 0; j < row; j++) { 21 printf("%d ", transpose[i][j]); 22 } 23 printf("\n"); 24 } 25 return 0; 26 } </pre> | Run | <pre> Enter the number of rows and columns: 2 2 Enter elements of the matrix: 1 2 3 4 Transpose of the matrix: 1 3 2 4 === Code Execution Successful === </pre> |

8.STACK OPERATIONS-(PUSH /POP/DISPLAY)

Programiz C Online Compiler

main.c

Share

Run

```
1 #include <stdio.h>
2 #define MAX 5
3 int stack[MAX], top = -1;
4 void push() {
5     int value;
6     if (top == MAX - 1) {
7         printf("Stack Overflow!\n");
8         return;
9     }
10    printf("Enter the value to push: ");
11    scanf("%d", &value);
12    stack[++top] = value;
13    printf("%d pushed onto the stack.\n", value);
14 }
15 void pop() {
16     if (top == -1) {
17         printf("Stack Underflow!\n");
18         return;
19     }
20    printf("%d popped from the stack.\n", stack[top--]);
21 }
22 void display() {
23     if (top == -1) {
24         printf("Stack is empty.\n");
25         return;
26     }
27    printf("Stack elements: ");
28    for (int i = top; i >= 0; i--) {
29        printf("%d ", stack[i]);
30    }
31    printf("\n");
32 }
33 int main() {
34     int choice;
35     while (1) {
36         printf("\nStack Operations:\n");
37         printf("1.Push\n2.Pop\n3.Display\n");
38         printf("Enter to print your choice: ");
39         scanf("%d", &choice);
40         switch (choice) {
41             case 1: push(); break;
42             case 2: pop(); break;
43             case 3: display(); break;
44             default: printf("Invalid choice!\n");
45         }
46     }
47     return 0;
48 }
49
50
```

Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice: 1
Enter the value to push: 44
44 pushed onto the stack.

Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice: 2
44 popped from the stack.

Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice: 3
Stack is empty.

Stack Operations:
1.Push
2.Pop
3.Display
Enter to print your choice:

9.QUEUE OPERATION-(INSERTION,DELETION,DISPLAY)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define SIZE 5
4  int queue[SIZE];
5  int front = -1;
6  int rear = -1;
7  int isFull() {
8      return (rear == SIZE - 1);
9  }
10 int isEmpty() {
11     return (front == -1);
12 }
13 void enqueue(int value) {
14     if (isFull()) {
15         printf("Queue is full! Insertion not possible.\n");
16     } else {
17         if (front == -1) {
18             front = 0;
19         }
20         rear++;
21         queue[rear] = value;
22         printf("Inserted %d into the queue.\n", value);
23     }
24 }
25 int dequeue() {
26     int value;
27     if (isEmpty()) {
28         printf("Queue is empty! Deletion not possible.\n");
29         return -1;
30     } else {
31         value = queue[front];
32         front++;
33         if (front > rear) {
34             front = rear = -1;
35         }
36         printf("Deleted %d from the queue.\n", value);
37         return value;
38     }
39 }
```

```
40 void display() {
41     if (isEmpty()) {
42         printf("Queue is empty!\n");
43     } else {
44         printf("Queue elements are: ");
45         for (int i = front; i <= rear; i++) {
46             printf("%d ", queue[i]);
47         }
48         printf("\n");
49     }
50 }
51 int main() {
52     int choice, value;
53     printf("Queue Operations:\n");
54     printf("1. Insert (Enqueue)\n");
55     printf("2. Delete (Dequeue)\n");
56     printf("3. Display\n");
57     printf("4. Exit\n");
58     do {
59         printf("Enter your choice: ");
60         scanf("%d", &choice);
61         switch (choice) {
62             case 1:
63                 printf("Enter value to insert: ");
64                 scanf("%d", &value);
65                 enqueue(value);
66                 break;
67             case 2:
68                 dequeue();
69                 break;
70             case 3:
71                 display();
72                 break;
73             case 4:
74                 printf("Exiting program.\n");
75                 break;
76             default:
77                 printf("Invalid choice! Please try again.\n");
78         }
79     } while (choice != 4);
80     return 0;
81 }
```

10.LINKED LIST OPERATION-(INSERTION,DELETION,DISPLAY)

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct linkedList {
4     int data;
5     struct linkedList *next;
6 };
7 struct linkedList *head = NULL;
8 void insert(int value) {
9     struct linkedList *newNode = (struct linkedList *)malloc(sizeof(struct linkedList));
10    newNode->data = value;
11    newNode->next = head;
12    head = newNode;
13    printf("Inserted %d\n", value);
14 }
15 void del() {
16     if (head == NULL) {
17         printf("Linked list is empty. Cannot delete.\n");
18         return;
19     }
20     struct linkedList *temp = head;
21     head = head->next;
22     printf("Deleted node with value: %d\n", temp->data);
23     free(temp);
24 }
25 void display() {
26     if (head == NULL) {
27         printf("Linked list is empty.\n");
28         return;
29     }
30     printf("Linked list: ");
31     struct linkedList *ptr = head;
32     while (ptr != NULL) {
33         printf("%d ", ptr->data);
34         ptr = ptr->next;
35     }
36     printf("\n");
37 }
38 int main() {
39     int choice, value;
40     printf("Linked List Operations:\n");
41     printf("1. Insert\n");
42     printf("2. Delete\n");
43     printf("3. Display\n");
44     printf("4. Exit\n");
45     do {
46         printf("Enter your choice: ");
47         scanf("%d", &choice);
48         switch (choice) {
49             case 1:
50                 printf("Enter value to insert: ");
51                 scanf("%d", &value);
52                 insert(value);
53                 break;
54             case 2:
55                 del();
56                 break;
57             case 3:
58                 display();
59                 break;
60             case 4:
61                 printf("Exiting program.\n");
62                 break;
63             default:
64                 printf("Invalid choice! Please try again.\n");
65         }
66     } while (choice != 4);
67     struct linkedList *current = head;
68     struct linkedList *next;
69     while (current != NULL) {
70         next = current->next;
71         free(current);
72         current = next;
73     }
74     head = NULL;
75     return 0;
76 }
```

11.stack through linked list


```

main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define MAX_SIZE 10
4 struct Stack {
5     int arr[MAX_SIZE];
6     int top;
7 };
8 void initializeStack(struct Stack *stack) {
9     stack->top = -1;
10 }
11 int isFull(struct Stack *stack) {
12     return stack->top == MAX_SIZE - 1;
13 }
14 int isEmpty(struct Stack *stack) {
15     return stack->top == -1;
16 }
17 void push(struct Stack *stack, int data) {
18     if (isFull(stack)) {
19         printf("Stack Overflow! Cannot push %d\n", data);
20     } else {
21         stack->arr[++stack->top] = data;
22         printf("%d pushed to stack\n", data);
23     }
24 }
25 int pop(struct Stack *stack) {
26     if (isEmpty(stack)) {
27         printf("Stack Underflow! Cannot pop\n");
28         return -1;
29     } else {
30         return stack->arr[stack->top--];
31     }
32 }
33 void display(struct Stack *stack) {
34     if (isEmpty(stack)) {
35         printf("Stack is empty.\n");
36     } else {
37         printf("Stack elements: ");
38         for (int i = stack->top; i >= 0; i--) {
39             printf("%d ", stack->arr[i]);
40         }
41         printf("\n");
42     }
43 }
44 int main() {
45     struct Stack stack;
46     initializeStack(&stack);
47     int choice, value;
48     printf("Stack Operations:\n");
49     printf("1. Push\n");
50     printf("2. Pop\n");
51     printf("3. Display\n");
52     printf("4. Exit\n");
53     do {
54         printf("Enter your choice: ");
55         scanf("%d", &choice);
56         switch (choice) {
57             case 1:
58                 printf("Enter value to push: ");
59                 scanf("%d", &value);
60                 push(&stack, value);
61                 break;
62             case 2:
63                 value = pop(&stack);
64                 if (value != -1) {
65                     printf("%d popped from stack\n", value);
66                 }
67                 break;
68             case 3:
69                 display(&stack);
70                 break;
71             case 4:
72                 printf("Exiting program.\n");
73                 break;
74             default:
75                 printf("Invalid choice! Please try again.\n");
76         }
77     } while (choice != 4);
78     return 0;
79 }

```

12.QUEUE THROUGH LINK LIST

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct QueueNode {
4     int data;
5     struct QueueNode *next;};
6 struct Queue {
7     struct QueueNode *front;
8     struct QueueNode *rear;};
9 struct QueueNode* createNode(int data) {
10     struct QueueNode* newNode = (struct QueueNode*)malloc(sizeof(struct QueueNode));
11     newNode->data = data;
12     newNode->next = NULL;
13     return newNode;
14 void initializeQueue(struct Queue *q) {
15     q->front = NULL;
16     q->rear = NULL;}
17 int isEmpty(struct Queue *q) {
18     return (q->front == NULL);}
19 void enqueue(struct Queue *q, int data) {
20     struct QueueNode* newNode = createNode(data);
21     if (isEmpty(q)) {
22         q->front = newNode;
23         q->rear = newNode;
24     } else {
25         q->rear->next = newNode;
26         q->rear = newNode;
27     printf("Enqueued %d\n", data);}
28 int dequeue(struct Queue *q) {
29     if (isEmpty(q)) {
30         printf("Queue Underflow! Cannot dequeue.\n");
31         return -1;
32     } else {
33         struct QueueNode* temp = q->front;
34         int dequeuedValue = temp->data;
35         q->front = q->front->next;
36         if (q->front == NULL) {
37             q->rear = NULL;
38         }
39         free(temp);
40         printf("Dequeued %d\n", dequeuedValue);
41         return dequeuedValue;
42     }
43 void display(struct Queue *q) {
44     if (isEmpty(q)) {
45         printf("Queue is empty.\n");
46     } else {
47         printf("Queue elements: ");
48         struct QueueNode* current = q->front;
49         while (current != NULL) {
50             printf("%d ", current->data);
51             current = current->next;
52         }
53         printf("\n");
54     }
55 }
56 int main() {
57     struct Queue q;
```

```

57     struct Queue q;
58     initializeQueue(&q);
59     int choice, value;
60     printf("Queue Operations (Linked List):\n");
61     printf("1. Enqueue (Insert)\n");
62     printf("2. Dequeue (Delete)\n");
63     printf("3. Display\n");
64     printf("4. Exit\n");
65     do {
66         printf("Enter your choice: ");
67         scanf("%d", &choice);
68         switch (choice) {
69             case 1:
70                 printf("Enter value to enqueue: ");
71                 scanf("%d", &value);
72                 enqueue(&q, value);
73                 break;
74             case 2:
75                 dequeue(&q);
76                 break;
77             case 3:
78                 display(&q);
79                 break;
80             case 4:
81                 printf("Exiting program.\n");
82                 break;
83             default:
84                 printf("Invalid choice! Please try again.\n");
85         }
86     } while (choice != 4);
87     struct QueueNode* current = q.front;
88     struct QueueNode* next;
89     while (current != NULL) {
90         next = current->next;
91         free(current);
92         current = next;
93     }
94     q.front = q.rear = NULL;
95     return 0;
96 }

```

13.TREE TRAVERSAL

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  struct Node {
4      int data;
5      struct Node *left;
6      struct Node *right;
7  };
8  struct Node* createNode(int data) {
9      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
10     newNode->data = data;
11     newNode->left = NULL;
12     newNode->right = NULL;
13     return newNode;
14 }
15 void inorder(struct Node* node) {
16     if (node == NULL)
17         return;
18     inorder(node->left);
19     printf("%d ", node->data);
20     inorder(node->right);
21 }
22 void preorder(struct Node* node) {
23     if (node == NULL)
24         return;
25     printf("%d ", node->data);
26     preorder(node->left);
27     preorder(node->right);
28 }
29 void postorder(struct Node* node) {
30     if (node == NULL)
31         return;
32     postorder(node->left);
33     postorder(node->right);
34     printf("%d ", node->data);
35 }
36 int main() {
37     struct Node* root = createNode(1);
38     root->left = createNode(2);
39     root->right = createNode(3);
40     root->left->left = createNode(4);
41     root->left->right = createNode(5);
42     printf("Inorder traversal: ");
43     inorder(root);
44     printf("\n");
45     printf("Preorder traversal: ");
46     preorder(root);
47     printf("\n");
48     printf("Postorder traversal: ");
49     postorder(root);
50     printf("\n");
51     return 0;
```

14.TREE SERCHING

```
main.c
1  #include <stdio.h>
2  int Search(int DATA[], int LB, int UB, int ITEM) {
3      int BEG = LB, END = UB, MID, LOC = -1;
4      while (BEG <= END) {
5          MID = (BEG + END) / 2;
6          if (ITEM == DATA[MID]) {
7              LOC = MID;
8              break;
9          } else if (ITEM < DATA[MID]) {
10             END = MID - 1;
11          } else {
12             BEG = MID + 1;
13          }
14      }
15      return LOC;
16  }
17  int main() {
18      int DATA[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
19      int n = sizeof(DATA) / sizeof(DATA[0]);
20      int ITEM, LOC;
21      printf("Item to search: ");
22      scanf("%d", &ITEM);
23      LOC = Search(DATA, 0, n - 1, ITEM);
24      if (LOC != -1) {
25          printf("Item found at location: %d\n", LOC + 1);
26      } else {
27          printf("Item not found in array\n");
28      }
29      return 0;
30  }
```

15.BINARY SEARCH

```

#include <stdio.h>
int binarySearch(int arr[], int n, int target) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

int main() {
    int n, target;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter %d sorted elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &target);
    int result = binarySearch(arr, n, target);
    if (result != -1) {
        printf("Element found at index: %d\n", result);
    } else {
        printf("Element not found.\n");
    }
    return 0;
}

```

16.SORTING USING QUICK SORT

main.c

```
1  #include <stdio.h>
2  void swap(int *a, int *b) {
3      int temp = *a;
4      *a = *b;
5      *b = temp;
6  }
7  int partition(int arr[], int low, int high) {
8      int pivot = arr[high];
9      int i = (low - 1);
10
11     for (int j = low; j <= high - 1; j++) {
12         if (arr[j] < pivot) {
13             i++;
14             swap(&arr[i], &arr[j]);
15         }
16     }
17     swap(&arr[i + 1], &arr[high]);
18     return (i + 1);
19 }
20 void quickSort(int arr[], int low, int high) {
21     if (low < high) {
22         int pi = partition(arr, low, high);
23         quickSort(arr, low, pi - 1);
24         quickSort(arr, pi + 1, high);
25     }
26 }
27 int main() {
28     int n;
29     printf("Enter number of elements: ");
30     scanf("%d", &n);
31     int arr[n];
32     printf("Enter %d elements:\n", n);
33     for (int i = 0; i < n; i++) {
34         scanf("%d", &arr[i]);
35     }
36     printf("Sorted array using Quick Sort: ");
37     quickSort(arr, 0, n - 1);
38     for (int i = 0; i < n; i++) {
39         printf("%d ", arr[i]);
40     }
41     printf("\n");
42     return 0;
43 }
```

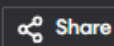
17.LINEAR SEARCH METHOD

main.c



```
1  #include <stdio.h>
2  int linearSearch(int arr[], int n, int target) {
3      for (int i = 0; i < n; i++) {
4          if (arr[i] == target) {
5              return i;
6          }
7      }
8      return -1;
9  }
10 int main() {
11     int n, target;
12     printf("Enter the number of elements: ");
13     scanf("%d", &n);
14     int arr[n];
15     printf("Enter %d elements:\n", n);
16     for (int i = 0; i < n; i++) {
17         scanf("%d", &arr[i]);
18     }
19     printf("Enter the element to search: ");
20     scanf("%d", &target);
21     int result = linearSearch(arr, n, target);
22     if (result != -1) {
23         printf("Element found at index: %d\n", result);
24     } else {
25         printf("Element not found in the array.\n");
26     }
27     return 0;
28 }
```

18.CONCATENATION OF TWO LINKED LIST



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node *next;
6 };
7 struct Node* head = NULL;
8 void insertAtBeginning(int data) {
9     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
10    newNode->data = data;
11    newNode->next = head;
12    head = newNode;
13    printf("Inserted %d at the beginning\n", data);
14 }
15 void insertAtEnd(int data) {
16     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
17     newNode->data = data;
18     newNode->next = NULL;
19     if (head == NULL) {
20         head = newNode;
21         printf("Inserted %d at the beginning (empty list)\n", data);
22         return;
23     }
24     struct Node* temp = head;
25     while (temp->next != NULL) {
26         temp = temp->next;
27     }
28     temp->next = newNode;
29     printf("Inserted %d at the end\n", data);
30 }
31 void deleteFromBeginning() {
32     if (head == NULL) {
33         printf("List is empty, cannot delete from beginning\n");
34         return;
35     }
36     struct Node* temp = head;
37     head = head->next;
38     printf("Deleted %d from the beginning\n", temp->data);
39     free(temp);
40 }
41 void deleteFromEnd() {
```

main.c



Share

Run

```
40 }
41 void deleteFromEnd() {
42     if (head == NULL) {
43         printf("List is empty, cannot delete from end\n");
44         return;
45     }
46     if (head->next == NULL) {
47         printf("Deleted %d from the end (single node)\n", head->data);
48         free(head);
49         head = NULL;
50         return;
51     }
52     struct Node* secondLast = head;
53     while (secondLast->next->next != NULL) {
54         secondLast = secondLast->next;
55     }
56     struct Node* lastNode = secondLast->next;
57     printf("Deleted %d from the end\n", lastNode->data);
58     secondLast->next = NULL;
59     free(lastNode);
60 }
61 void displayList() {
62     struct Node* temp = head;
63     printf("List elements: ");
64     while (temp != NULL) {
65         printf("%d ", temp->data);
66         temp = temp->next;
67     }
68     printf("\n");
69 }
70 int main() {
71     insertAtBeginning(10);
72     insertAtEnd(20);
73     insertAtBeginning(5);
74     insertAtEnd(30);
75     displayList();
76     deleteFromBeginning();
77     deleteFromEnd();
78     displayList();
79     return 0;
80 }
```

19.REVERSE A LINKED LIST

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 struct Node {
4     int data;
5     struct Node *next;
6 };
7 struct Node* reverseLinkedList(struct Node* head) {
8     struct Node *prev = NULL;
9     struct Node *current = head;
10    struct Node *next = NULL;
11    while (current != NULL) {
12        next = current->next;
13        current->next = prev;
14        prev = current;
15        current = next;
16    }
17    return prev;
18 }
19 void append(struct Node** head_ref, int new_data) {
20     struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
21     struct Node *last = *head_ref;
22     new_node->data = new_data;
23     new_node->next = NULL;
24     if (*head_ref == NULL) {
25         *head_ref = new_node;
26         return;
27     }
28     while (last->next != NULL)
29         last = last->next;
30     last->next = new_node;
31 }
32 void printList(struct Node *node) {
33     printf("Linked List: ");
34     while (node != NULL) {
35         printf("%d -> ", node->data);
36         node = node->next;
37     }
38     printf("NULL\n");
39 }
40 int main() {
41     struct Node* head = NULL;
42     int n, value;
43     printf("Enter the number of nodes: ");
44     scanf("%d", &n);
45     printf("Enter %d node values:\n", n);
46     for (int i = 0; i < n; i++) {
47         scanf("%d", &value);
48         append(&head, value);
49     }
50     printf("Original Linked List: ");
51     printList(head);
52     head = reverseLinkedList(head);
53     printf("Reversed Linked List: ");
54     printList(head);
55     struct Node* current = head;
56     struct Node* next;
57     while (current != NULL) {
58         next = current->next;
59         free(current);
60         current = next;
61     }
62     head = NULL;
63     return 0;
64 }

```

20.SORTING USING MERGE SORT

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 void merge(int arr[], int l, int m, int r) {
4     int i, j, k;
5     int n1 = m - l + 1;
6     int n2 = r - m;
7     int L[n1], R[n2];
8     for (i = 0; i < n1; i++)
9         L[i] = arr[l + i];
10    for (j = 0; j < n2; j++)
11        R[j] = arr[m + 1 + j];
12    i = 0;
13    j = 0;
14    k = l;
15    while (i < n1 && j < n2) {
16        if (L[i] <= R[j]) {
17            arr[k] = L[i];
18            i++;
19        } else {
20            arr[k] = R[j];
21            j++;
22        }
23        k++;
24    }
25    while (i < n1) {
26        arr[k] = L[i];
27        i++;
28        k++;
29    }
30    while (j < n2) {
31        arr[k] = R[j];
32        j++;
33        k++;
34    }
35 }
36 void mergeSort(int arr[], int l, int r) {
37     if (l < r) {
38         int m = l + (r - l) / 2;
39         mergeSort(arr, l, m);
40         mergeSort(arr, m + 1, r);
41         merge(arr, l, m, r);
42     }
43 }
44 void printArray(int arr[], int size) {
45     int i;
46     for (i = 0; i < size; i++)
47         printf("%d ", arr[i]);
48     printf("\n");
49 }
50 int main() {
51     int n;
52     printf("Enter number of elements: ");
53     scanf("%d", &n);
54     int arr[n];
55     printf("Enter %d elements:\n", n);
56     for (int i = 0; i < n; i++) {
57         scanf("%d", &arr[i]);
58     }
59     printf("Original array: ");
60     printArray(arr, n);
61     mergeSort(arr, 0, n - 1);
62     printf("Sorted array using Merge Sort: ");
63     printArray(arr, n);
64     return 0;
65 }

```