Array implementation of stack

1. #include <stdio.h>
2. int stack[100],i,j,choice=0,n,top=-1;
3. void push();
4. void pop();
5. void show();
6. void main ()
7. {
9. printf("Enter the number of elements in the stack ");
10. scanf("%d",&n);
11. printf("\*\*\*\*\*\*\*\*\*Stack operations using array\*\*\*\*\*\*\*\*\*");
13. printf("\n----------------------------------------------\n");
14. while(choice != 4)
15. {
16. printf("Chose one from the below options...\n");
17. printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
18. printf("\n Enter your choice \n");
19. scanf("%d",&choice);
20. switch(choice)
21. {
22. case 1:
23. {
24. push();
25. break;
26. }
27. case 2:
28. {
29. pop();
30. break;
31. }
32. case 3:
33. {
34. show();
35. break;
36. }
37. case 4:
38. {
39. printf("Exiting....");
40. break;
41. }
42. default:
43. {
44. printf("Please Enter valid choice ");
45. }
46. };
47. }
48. }
50. void push ()
51. {
52. int val;
53. if (top == n )
54. printf("\n Overflow");
55. else
56. {
57. printf("Enter the value?");
58. scanf("%d",&val);
59. top = top +1;
60. stack[top] = val;
61. }
62. }
64. void pop ()
65. {
66. if(top == -1)
67. printf("Underflow");
68. else
69. top = top -1;
70. }
71. void show()
72. {
73. for (i=top;i>=0;i--)
74. {
75. printf("%d\n",stack[i]);
76. }
77. if(top == -1)
78. {
79. printf("Stack is empty");
80. }
81. }

**Java Program**

1. import java.util.Scanner;
2. class Stack
3. {
4. int top;
5. int maxsize = 10;
6. int[] arr = new int[maxsize];

9. boolean isEmpty()
10. {
11. return (top < 0);
12. }
13. Stack()
14. {
15. top = -1;
16. }
17. boolean push (Scanner sc)
18. {
19. if(top == maxsize-1)
20. {
21. System.out.println("Overflow !!");
22. return false;
23. }
24. else
25. {
26. System.out.println("Enter Value");
27. int val = sc.nextInt();
28. top++;
29. arr[top]=val;
30. System.out.println("Item pushed");
31. return true;
32. }
33. }
34. boolean pop ()
35. {
36. if (top == -1)
37. {
38. System.out.println("Underflow !!");
39. return false;
40. }
41. else
42. {
43. top --;
44. System.out.println("Item popped");
45. return true;
46. }
47. }
48. void display ()
49. {
50. System.out.println("Printing stack elements .....");
51. for(int i = top; i>=0;i--)
52. {
53. System.out.println(arr[i]);
54. }
55. }
56. }
57. public class Stack\_Operations {
58. public static void main(String[] args) {
59. int choice=0;
60. Scanner sc = new Scanner(System.in);
61. Stack s = new Stack();
62. System.out.println("\*\*\*\*\*\*\*\*\*Stack operations using array\*\*\*\*\*\*\*\*\*\n");
63. System.out.println("\n------------------------------------------------\n");
64. while(choice != 4)
65. {
66. System.out.println("\nChose one from the below options...\n");
67. System.out.println("\n1.Push\n2.Pop\n3.Show\n4.Exit");
68. System.out.println("\n Enter your choice \n");
69. choice = sc.nextInt();
70. switch(choice)
71. {
72. case 1:
73. {
74. s.push(sc);
75. break;
76. }
77. case 2:
78. {
79. s.pop();
80. break;
81. }
82. case 3:
83. {
84. s.display();
85. break;
86. }
87. case 4:
88. {
89. System.out.println("Exiting....");
90. System.exit(0);
91. break;
92. }
93. default:
94. {
95. System.out.println("Please Enter valid choice ");
96. }
97. };
98. }
99. }
100. }

Linked List implementation of stack:

1. #include <stdio.h>
2. #include <stdlib.h>
3. void push();
4. void pop();
5. void display();
6. struct node
7. {
8. int val;
9. struct node \*next;
10. };
11. struct node \*head;
13. void main ()
14. {
15. int choice=0;
16. printf("\n\*\*\*\*\*\*\*\*\*Stack operations using linked list\*\*\*\*\*\*\*\*\*\n");
17. printf("\n----------------------------------------------\n");
18. while(choice != 4)
19. {
20. printf("\n\nChose one from the below options...\n");
21. printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
22. printf("\n Enter your choice \n");
23. scanf("%d",&choice);
24. switch(choice)
25. {
26. case 1:
27. {
28. push();
29. break;
30. }
31. case 2:
32. {
33. pop();
34. break;
35. }
36. case 3:
37. {
38. display();
39. break;
40. }
41. case 4:
42. {
43. printf("Exiting....");
44. break;
45. }
46. default:
47. {
48. printf("Please Enter valid choice ");
49. }
50. };
51. }
52. }
53. void push ()
54. {
55. int val;
56. struct node \*ptr = (struct node\*)malloc(sizeof(struct node));
57. if(ptr == NULL)
58. {
59. printf("not able to push the element");
60. }
61. else
62. {
63. printf("Enter the value");
64. scanf("%d",&val);
65. if(head==NULL)
66. {
67. ptr->val = val;
68. ptr -> next = NULL;
69. head=ptr;
70. }
71. else
72. {
73. ptr->val = val;
74. ptr->next = head;
75. head=ptr;
77. }
78. printf("Item pushed");
80. }
81. }
83. void pop()
84. {
85. int item;
86. struct node \*ptr;
87. if (head == NULL)
88. {
89. printf("Underflow");
90. }
91. else
92. {
93. item = head->val;
94. ptr = head;
95. head = head->next;
96. free(ptr);
97. printf("Item popped");
99. }
100. }
101. void display()
102. {
103. int i;
104. struct node \*ptr;
105. ptr=head;
106. if(ptr == NULL)
107. {
108. printf("Stack is empty\n");
109. }
110. else
111. {
112. printf("Printing Stack elements \n");
113. while(ptr!=NULL)
114. {
115. printf("%d\n",ptr->val);
116. ptr = ptr->next;
117. }
118. }
119. }

Queue:

Array representation of Queue:

1. #include<stdio.h>
2. #include<stdlib.h>
3. #define maxsize 5
4. void insert();
5. void delete();
6. void display();
7. int front = -1, rear = -1;
8. int queue[maxsize];
9. void main ()
10. {
11. int choice;
12. while(choice != 4)
13. {
14. printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
15. printf("\n=================================================================\n");
16. printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
17. printf("\nEnter your choice ?");
18. scanf("%d",&choice);
19. switch(choice)
20. {
21. case 1:
22. insert();
23. break;
24. case 2:
25. delete();
26. break;
27. case 3:
28. display();
29. break;
30. case 4:
31. exit(0);
32. break;
33. default:
34. printf("\nEnter valid choice??\n");
35. }
36. }
37. }
38. void insert()
39. {
40. int item;
41. printf("\nEnter the element\n");
42. scanf("\n%d",&item);
43. if(rear == maxsize-1)
44. {
45. printf("\nOVERFLOW\n");
46. return;
47. }
48. if(front == -1 && rear == -1)
49. {
50. front = 0;
51. rear = 0;
52. }
53. else
54. {
55. rear = rear+1;
56. }
57. queue[rear] = item;
58. printf("\nValue inserted ");
60. }
61. void delete()
62. {
63. int item;
64. if (front == -1 || front > rear)
65. {
66. printf("\nUNDERFLOW\n");
67. return;
69. }
70. else
71. {
72. item = queue[front];
73. if(front == rear)
74. {
75. front = -1;
76. rear = -1 ;
77. }
78. else
79. {
80. front = front + 1;
81. }
82. printf("\nvalue deleted ");
83. }

86. }
88. void display()
89. {
90. int i;
91. if(rear == -1)
92. {
93. printf("\nEmpty queue\n");
94. }
95. else
96. {   printf("\nprinting values .....\n");
97. for(i=front;i<=rear;i++)
98. {
99. printf("\n%d\n",queue[i]);
100. }
101. }
102. }

**Output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*

==============================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

123

Value inserted

\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*

==============================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

90

Value inserted

\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*

===================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

value deleted

\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*

==============================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

90

\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*

==============================================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?4

Linked List implementation of Queue:

1. #include<stdio.h>
2. #include<stdlib.h>
3. struct node
4. {
5. int data;
6. struct node \*next;
7. };
8. struct node \*front;
9. struct node \*rear;
10. void insert();
11. void delete();
12. void display();
13. void main ()
14. {
15. int choice;
16. while(choice != 4)
17. {
18. printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");
19. printf("\n=================================================================\n");
20. printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");
21. printf("\nEnter your choice ?");
22. scanf("%d",& choice);
23. switch(choice)
24. {
25. case 1:
26. insert();
27. break;
28. case 2:
29. delete();
30. break;
31. case 3:
32. display();
33. break;
34. case 4:
35. exit(0);
36. break;
37. default:
38. printf("\nEnter valid choice??\n");
39. }
40. }
41. }
42. void insert()
43. {
44. struct node \*ptr;
45. int item;
47. ptr = (struct node \*) malloc (sizeof(struct node));
48. if(ptr == NULL)
49. {
50. printf("\nOVERFLOW\n");
51. return;
52. }
53. else
54. {
55. printf("\nEnter value?\n");
56. scanf("%d",&item);
57. ptr -> data = item;
58. if(front == NULL)
59. {
60. front = ptr;
61. rear = ptr;
62. front -> next = NULL;
63. rear -> next = NULL;
64. }
65. else
66. {
67. rear -> next = ptr;
68. rear = ptr;
69. rear->next = NULL;
70. }
71. }
72. }
73. void delete ()
74. {
75. struct node \*ptr;
76. if(front == NULL)
77. {
78. printf("\nUNDERFLOW\n");
79. return;
80. }
81. else
82. {
83. ptr = front;
84. front = front -> next;
85. free(ptr);
86. }
87. }
88. void display()
89. {
90. struct node \*ptr;
91. ptr = front;
92. if(front == NULL)
93. {
94. printf("\nEmpty queue\n");
95. }
96. else
97. {   printf("\nprinting values .....\n");
98. while(ptr != NULL)
99. {
100. printf("\n%d\n",ptr -> data);
101. ptr = ptr -> next;
102. }
103. }
104. }

**Output:**

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

123

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

90

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

123

90

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

90

\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*

==============================

1.insert an element

2.Delete an element

3.Display the queue

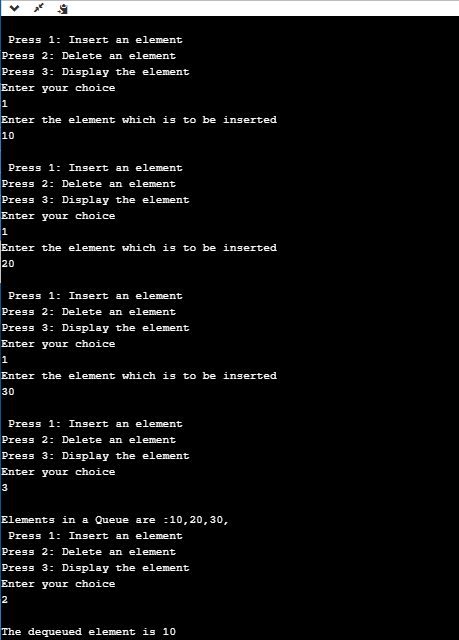
4.Exit

Enter your choice ?4

**Implementation of circular queue using Array**

1. #include <stdio.h>
3. # define max 6
4. int queue[max];  // array declaration
5. int front=-1;
6. int rear=-1;
7. // function to insert an element in a circular queue
8. void enqueue(int element)
9. {
10. if(front==-1 && rear==-1)   // condition to check queue is empty
11. {
12. front=0;
13. rear=0;
14. queue[rear]=element;
15. }
16. else if((rear+1)%max==front)  // condition to check queue is full
17. {
18. printf("Queue is overflow..");
19. }
20. else
21. {
22. rear=(rear+1)%max;       // rear is incremented
23. queue[rear]=element;     // assigning a value to the queue at the rear position.
24. }
25. }
27. // function to delete the element from the queue
28. int dequeue()
29. {
30. if((front==-1) && (rear==-1))  // condition to check queue is empty
31. {
32. printf("\nQueue is underflow..");
33. }
34. else if(front==rear)
35. {
36. printf("\nThe dequeued element is %d", queue[front]);
37. front=-1;
38. rear=-1;
39. }
40. else
41. {
42. printf("\nThe dequeued element is %d", queue[front]);
43. front=(front+1)%max;
44. }
45. }
46. // function to display the elements of a queue
47. void display()
48. {
49. int i=front;
50. if(front==-1 && rear==-1)
51. {
52. printf("\n Queue is empty..");
53. }
54. else
55. {
56. printf("\nElements in a Queue are :");
57. while(i<=rear)
58. {
59. printf("%d,", queue[i]);
60. i=(i+1)%max;
61. }
62. }
63. }
64. int main()
65. {
66. int choice=1,x;   // variables declaration
68. while(choice<4 && choice!=0)   // while loop
69. {
70. printf("\n Press 1: Insert an element");
71. printf("\nPress 2: Delete an element");
72. printf("\nPress 3: Display the element");
73. printf("\nEnter your choice");
74. scanf("%d", &choice);
76. switch(choice)
77. {
79. case 1:
81. printf("Enter the element which is to be inserted");
82. scanf("%d", &x);
83. enqueue(x);
84. break;
85. case 2:
86. dequeue();
87. break;
88. case 3:
89. display();
91. }}
92. return 0;
93. }

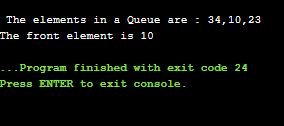
**Output:**



### Implementation of circular queue using linked list

1. #include <stdio.h>
2. // Declaration of struct type node
3. struct node
4. {
5. int data;
6. struct node \*next;
7. };
8. struct node \*front=-1;
9. struct node \*rear=-1;
10. // function to insert the element in the Queue
11. void enqueue(int x)
12. {
13. struct node \*newnode;  // declaration of pointer of struct node type.
14. newnode=(struct node \*)malloc(sizeof(struct node));  // allocating the memory to the newnode
15. newnode->data=x;
16. newnode->next=0;
17. if(rear==-1)  // checking whether the Queue is empty or not.
18. {
19. front=rear=newnode;
20. rear->next=front;
21. }
22. else
23. {
24. rear->next=newnode;
25. rear=newnode;
26. rear->next=front;
27. }
28. }
30. // function to delete the element from the queue
31. void dequeue()
32. {
33. struct node \*temp;   // declaration of pointer of node type
34. temp=front;
35. if((front==-1)&&(rear==-1))  // checking whether the queue is empty or not
36. {
37. printf("\nQueue is empty");
38. }
39. else if(front==rear)  // checking whether the single element is left in the queue
40. {
41. front=rear=-1;
42. free(temp);
43. }
44. else
45. {
46. front=front->next;
47. rear->next=front;
48. free(temp);
49. }
50. }
52. // function to get the front of the queue
53. int peek()
54. {
55. if((front==-1) &&(rear==-1))
56. {
57. printf("\nQueue is empty");
58. }
59. else
60. {
61. printf("\nThe front element is %d", front->data);
62. }
63. }
65. // function to display all the elements of the queue
66. void display()
67. {
68. struct node \*temp;
69. temp=front;
70. printf("\n The elements in a Queue are : ");
71. if((front==-1) && (rear==-1))
72. {
73. printf("Queue is empty");
74. }
76. else
77. {
78. while(temp->next!=front)
79. {
80. printf("%d,", temp->data);
81. temp=temp->next;
82. }
83. printf("%d", temp->data);
84. }
85. }
87. void main()
88. {
89. enqueue(34);
90. enqueue(10);
91. enqueue(23);
92. display();
93. dequeue();
94. peek();
95. }

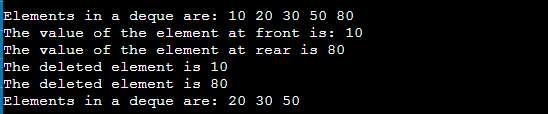
**Output:**



# Deque (or double-ended queue)

1. #include <stdio.h>
2. #define size 5
3. int deque[size];
4. int f = -1, r = -1;
5. //  insert\_front function will insert the value from the front
6. void insert\_front(int x)
7. {
8. if((f==0 && r==size-1) || (f==r+1))
9. {
10. printf("Overflow");
11. }
12. else if((f==-1) && (r==-1))
13. {
14. f=r=0;
15. deque[f]=x;
16. }
17. else if(f==0)
18. {
19. f=size-1;
20. deque[f]=x;
21. }
22. else
23. {
24. f=f-1;
25. deque[f]=x;
26. }
27. }
29. // insert\_rear function will insert the value from the rear
30. void insert\_rear(int x)
31. {
32. if((f==0 && r==size-1) || (f==r+1))
33. {
34. printf("Overflow");
35. }
36. else if((f==-1) && (r==-1))
37. {
38. r=0;
39. deque[r]=x;
40. }
41. else if(r==size-1)
42. {
43. r=0;
44. deque[r]=x;
45. }
46. else
47. {
48. r++;
49. deque[r]=x;
50. }
52. }
54. // display function prints all the value of deque.
55. void display()
56. {
57. int i=f;
58. printf("\nElements in a deque are: ");
60. while(i!=r)
61. {
62. printf("%d ",deque[i]);
63. i=(i+1)%size;
64. }
65. printf("%d",deque[r]);
66. }
68. // getfront function retrieves the first value of the deque.
69. void getfront()
70. {
71. if((f==-1) && (r==-1))
72. {
73. printf("Deque is empty");
74. }
75. else
76. {
77. printf("\nThe value of the element at front is: %d", deque[f]);
78. }
80. }
82. // getrear function retrieves the last value of the deque.
83. void getrear()
84. {
85. if((f==-1) && (r==-1))
86. {
87. printf("Deque is empty");
88. }
89. else
90. {
91. printf("\nThe value of the element at rear is %d", deque[r]);
92. }
94. }
96. // delete\_front() function deletes the element from the front
97. void delete\_front()
98. {
99. if((f==-1) && (r==-1))
100. {
101. printf("Deque is empty");
102. }
103. else if(f==r)
104. {
105. printf("\nThe deleted element is %d", deque[f]);
106. f=-1;
107. r=-1;
109. }
110. else if(f==(size-1))
111. {
112. printf("\nThe deleted element is %d", deque[f]);
113. f=0;
114. }
115. else
116. {
117. printf("\nThe deleted element is %d", deque[f]);
118. f=f+1;
119. }
120. }
122. // delete\_rear() function deletes the element from the rear
123. void delete\_rear()
124. {
125. if((f==-1) && (r==-1))
126. {
127. printf("Deque is empty");
128. }
129. else if(f==r)
130. {
131. printf("\nThe deleted element is %d", deque[r]);
132. f=-1;
133. r=-1;
135. }
136. else if(r==0)
137. {
138. printf("\nThe deleted element is %d", deque[r]);
139. r=size-1;
140. }
141. else
142. {
143. printf("\nThe deleted element is %d", deque[r]);
144. r=r-1;
145. }
146. }
148. int main()
149. {
150. insert\_front(20);
151. insert\_front(10);
152. insert\_rear(30);
153. insert\_rear(50);
154. insert\_rear(80);
155. display();  // Calling the display function to retrieve the values of deque
156. getfront();  // Retrieve the value at front-end
157. getrear();  // Retrieve the value at rear-end
158. delete\_front();
159. delete\_rear();
160. display(); // calling display function to retrieve values after deletion
161. return 0;
162. }

**Output:**



1. #include <stdio.h>
2. #define size 5
3. int deque[size];
4. int f = -1, r = -1;
5. //  insert\_front function will insert the value from the front
6. void insert\_front(int x)
7. {
8. if((f==0 && r==size-1) || (f==r+1))
9. {
10. printf("Overflow");
11. }
12. else if((f==-1) && (r==-1))
13. {
14. f=r=0;
15. deque[f]=x;
16. }
17. else if(f==0)
18. {
19. f=size-1;
20. deque[f]=x;
21. }
22. else
23. {
24. f=f-1;
25. deque[f]=x;
26. }
27. }
29. // insert\_rear function will insert the value from the rear
30. void insert\_rear(int x)
31. {
32. if((f==0 && r==size-1) || (f==r+1))
33. {
34. printf("Overflow");
35. }
36. else if((f==-1) && (r==-1))
37. {
38. r=0;
39. deque[r]=x;
40. }
41. else if(r==size-1)
42. {
43. r=0;
44. deque[r]=x;
45. }
46. else
47. {
48. r++;
49. deque[r]=x;
50. }
52. }
54. // display function prints all the value of deque.
55. void display()
56. {
57. int i=f;
58. printf("\nElements in a deque are: ");
60. while(i!=r)
61. {
62. printf("%d ",deque[i]);
63. i=(i+1)%size;
64. }
65. printf("%d",deque[r]);
66. }
68. // getfront function retrieves the first value of the deque.
69. void getfront()
70. {
71. if((f==-1) && (r==-1))
72. {
73. printf("Deque is empty");
74. }
75. else
76. {
77. printf("\nThe value of the element at front is: %d", deque[f]);
78. }
80. }
82. // getrear function retrieves the last value of the deque.
83. void getrear()
84. {
85. if((f==-1) && (r==-1))
86. {
87. printf("Deque is empty");
88. }
89. else
90. {
91. printf("\nThe value of the element at rear is %d", deque[r]);
92. }
94. }
96. // delete\_front() function deletes the element from the front
97. void delete\_front()
98. {
99. if((f==-1) && (r==-1))
100. {
101. printf("Deque is empty");
102. }
103. else if(f==r)
104. {
105. printf("\nThe deleted element is %d", deque[f]);
106. f=-1;
107. r=-1;
109. }
110. else if(f==(size-1))
111. {
112. printf("\nThe deleted element is %d", deque[f]);
113. f=0;
114. }
115. else
116. {
117. printf("\nThe deleted element is %d", deque[f]);
118. f=f+1;
119. }
120. }
122. // delete\_rear() function deletes the element from the rear
123. void delete\_rear()
124. {
125. if((f==-1) && (r==-1))
126. {
127. printf("Deque is empty");
128. }
129. else if(f==r)
130. {
131. printf("\nThe deleted element is %d", deque[r]);
132. f=-1;
133. r=-1;
135. }
136. else if(r==0)
137. {
138. printf("\nThe deleted element is %d", deque[r]);
139. r=size-1;
140. }
141. else
142. {
143. printf("\nThe deleted element is %d", deque[r]);
144. r=r-1;
145. }
146. }
148. int main()
149. {
150. insert\_front(20);
151. insert\_front(10);
152. insert\_rear(30);
153. insert\_rear(50);
154. insert\_rear(80);
155. display();  // Calling the display function to retrieve the values of deque
156. getfront();  // Retrieve the value at front-end
157. getrear();  // Retrieve the value at rear-end
158. delete\_front();
159. delete\_rear();
160. display(); // calling display function to retrieve values after deletion
161. return 0;
162. }

**Output:**

Linked List implenenta