LAB EXERCISE 3

QUESTION 1 (CIRCULAR DOUBLY LINKED LIST)

```
1. /*
2.  * C++ Program to Implement Circular Doubly Linked List
3.  */
4. #include<iostream>
5. #include<cstdio>
6. #include<cstdlib>
7. using namespace std;
8.
9. /*
10.  * Node Declaration
11.  */
12. struct node
13. {
14.     int info;
15.     struct node *next;
16.     struct node *prev;
17. }*start, *last;
18. int counter = 0;
19. /*
20.  * Class Declaration
21.  */
22. class double_clist
23. {
24.     public:
25.         node *create_node(int);
26.         void insert_begin();
27.         void insert_last();
28.         void insert_pos();
29.         void delete_pos();
30.         void search();
31.         void update();
32.         void display();
33.         void reverse();
34.         void sort();
35.         double_clist()
36.         {
37.             start = NULL;
38.             last = NULL;
39.         }
40. };
41.
42. /*
```

```
 * Main: Contains Menu
 */
int main()
{
    int choice;
    double_clist cdl;
    while (1)
    {
        cout<<"\n--------------------------------"<<endl;
        cout<<"Operations on Doubly Circular linked list"<<endl;
        cout<<"\n--------------------------------"<<endl;
        cout<<"1.Insert at Beginning"<<endl;
        cout<<"2.Insert at Last"<<endl;
        cout<<"3.Insert at Position"<<endl;
        cout<<"4.Delete at Position"<<endl;
        cout<<"5.Update Node"<<endl;
        cout<<"6.Search Element"<<endl;
        cout<<"7.Sort"<<endl;
        cout<<"8.Display List"<<endl;
        cout<<"9.Reverse List"<<endl;
        cout<<"10.Exit"<<endl;
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            cdl.insert_begin();
            break;
        case 2:
            cdl.insert_last();
            break;
        case 3:
            cdl.insert_pos();
            break;
        case 4:
            cdl.delete_pos();
            break;
        case 5:
            cdl.update();
            break;
        case 6:
            cdl.search();
            break;
        case 7:
            cdl.sort();
            break;
```

```
89.         case 8:
90.             cdl.display();
91.             break;
92.         case 9:
93.             cdl.reverse();
94.             break;
95.         case 10:
96.             exit(1);
97.         default:
98.             cout<<"Wrong choice"<<endl;
99.         }
100.            }
101.            return 0;
102.        }
103.
104.        /*
105.         *MEMORY ALLOCATED FOR NODE DYNAMICALLY
106.         */
107.        node* double_clist::create_node(int value)
108.        {
109.            counter++;
110.            struct node *temp;
111.            temp = new(struct node);
112.            temp->info = value;
113.            temp->next = NULL;
114.            temp->prev = NULL;
115.            return temp;
116.        }
117.        /*
118.         *INSERTS ELEMENT AT BEGINNING
119.         */
120.        void double_clist::insert_begin()
121.        {
122.            int value;
123.            cout<<endl<<"Enter the element to be inserted: ";
124.            cin>>value;
125.            struct node *temp;
126.            temp = create_node(value);
127.            if (start == last && start == NULL)
128.            {
129.                cout<<"Element inserted in empty list"<<endl;
130.                start = last = temp;
131.                start->next = last->next = NULL;
132.                start->prev = last->prev = NULL;
133.            }
134.            else
```

```
135.            {
136.                temp->next = start;
137.                start->prev = temp;
138.                start = temp;
139.                start->prev = last;
140.                last->next = start;
141.                cout<<"Element inserted"<<endl;
142.            }
143.        }
144.
145.        /*
146.         *INSERTS ELEMNET AT LAST
147.         */
148.        void double_clist::insert_last()
149.        {
150.            int value;
151.            cout<<endl<<"Enter the element to be inserted: ";
152.            cin>>value;
153.            struct node *temp;
154.            temp = create_node(value);
155.            if (start == last && start == NULL)
156.            {
157.                cout<<"Element inserted in empty list"<<endl;
158.                start = last = temp;
159.                start->next = last->next = NULL;
160.                start->prev = last->prev = NULL;
161.            }
162.            else
163.            {
164.                last->next = temp;
165.                temp->prev = last;
166.                last = temp;
167.                start->prev = last;
168.                last->next = start;
169.            }
170.        }
171.        /*
172.         *INSERTS ELEMENT AT POSITION
173.         */
174.        void double_clist::insert_pos()
175.        {
176.            int value, pos, i;
177.            cout<<endl<<"Enter the element to be inserted: ";
178.            cin>>value;
179.            cout<<endl<<"Enter the postion of element inserted: ";
180.            cin>>pos;
```

```
181.            struct node *temp, *s, *ptr;
182.            temp = create_node(value);
183.            if (start == last && start == NULL)
184.            {
185.                if (pos == 1)
186.                {
187.                    start = last = temp;
188.                    start->next = last->next = NULL;
189.                    start->prev = last->prev = NULL;
190.                }
191.                else
192.                {
193.                    cout<<"Position out of range"<<endl;
194.                    counter--;
195.                    return;
196.                }
197.            }
198.            else
199.            {
200.                if (counter < pos)
201.                {
202.                     cout<<"Position out of range"<<endl;
203.                     counter--;
204.                     return;
205.                }
206.                s = start;
207.                for (i = 1;i <= counter;i++)
208.                {
209.                    ptr = s;
210.                    s = s->next;
211.                    if (i == pos - 1)
212.                    {
213.                        ptr->next = temp;
214.                        temp->prev = ptr;
215.                        temp->next = s;
216.                        s->prev = temp;
217.                        cout<<"Element inserted"<<endl;
218.                        break;
219.                    }
220.                }
221.            }
222.        }
223.        /*
224.         * Delete Node at Particular Position
225.         */
226.        void double_clist::delete_pos()
```

```
227.        {
228.            int pos, i;
229.            node *ptr, *s;
230.            if (start == last && start == NULL)
231.            {
232.                cout<<"List is empty, nothing to delete"<<endl;
233.                return;
234.            }
235.            cout<<endl<<"Enter the postion of element to be deleted: ";
236.            cin>>pos;
237.            if (counter < pos)
238.            {
239.                cout<<"Position out of range"<<endl;
240.                return;
241.            }
242.            s = start;
243.            if(pos == 1)
244.            {
245.                counter--;
246.                last->next = s->next;
247.                s->next->prev = last;
248.                start = s->next;
249.                free(s);
250.                cout<<"Element Deleted"<<endl;
251.                return;
252.            }
253.            for (i = 0;i < pos - 1;i++ )
254.            {
255.                s = s->next;
256.                ptr = s->prev;
257.            }
258.            ptr->next = s->next;
259.            s->next->prev = ptr;
260.            if (pos == counter)
261.            {
262.                last = ptr;
263.            }
264.            counter--;
265.            free(s);
266.            cout<<"Element Deleted"<<endl;
267.        }
268.        /*
269.         * Update value of a particular node
270.         */
271.        void double_clist::update()
272.        {
```

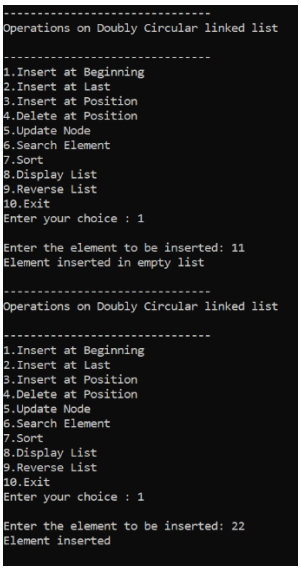
```
273.            int value, i, pos;
274.            if (start == last && start == NULL)
275.            {
276.                cout<<"The List is empty, nothing to update"<<endl;
277.                return;
278.            }
279.            cout<<endl<<"Enter the postion of node to be updated: ";
280.            cin>>pos;
281.            cout<<"Enter the new value: ";
282.            cin>>value;
283.            struct node *s;
284.            if (counter < pos)
285.            {
286.                cout<<"Position out of range"<<endl;
287.                return;
288.            }
289.            s = start;
290.            if (pos == 1)
291.            {
292.               s->info = value;
293.               cout<<"Node Updated"<<endl;
294.               return;
295.            }
296.            for (i=0;i < pos - 1;i++)
297.            {
298.                s = s->next;
299.            }
300.            s->info = value;
301.            cout<<"Node Updated"<<endl;
302.        }
303.        /*
304.         * Search Element in the list
305.         */
306.        void double_clist::search()
307.        {
308.            int pos = 0, value, i;
309.            bool flag = false;
310.            struct node *s;
311.            if (start == last && start == NULL)
312.            {
313.                cout<<"The List is empty, nothing to search"<<endl;
314.                return;
315.            }
316.            cout<<endl<<"Enter the value to be searched: ";
317.            cin>>value;
318.            s = start;
```

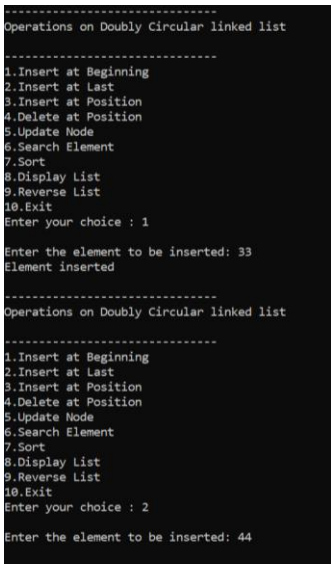
```
            for (i = 0;i < counter;i++)
            {
                pos++;
                if (s->info == value)
                {
                    cout<<"Element "<<value<<" found at position: "<<pos<<endl;
                    flag = true;
                }
                s = s->next;
            }
            if (!flag)
                cout<<"Element not found in the list"<<endl;
        }
        /*
         * Sorting Doubly Circular Link List
         */
        void double_clist::sort()
        {
            struct node *temp, *s;
            int value, i;
            if (start == last && start == NULL)
            {
                cout<<"The List is empty, nothing to sort"<<endl;
                return;
            }
            s = start;
            for (i = 0;i < counter;i++)
            {
                temp = s->next;
                while (temp != start)
                {
                    if (s->info > temp->info)
                    {
                        value = s->info;
                        s->info = temp->info;
                        temp->info = value;
                    }
                    temp = temp->next;
                }
                s = s->next;
            }
        }
        /*
         * Display Elements of the List
         */
        void double_clist::display()
```

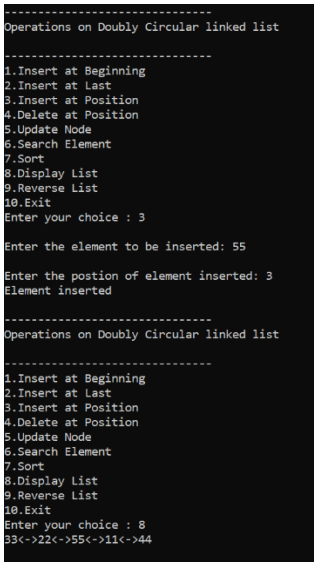
```
365.        {
366.            int i;
367.            struct node *s;
368.            if (start == last && start == NULL)
369.            {
370.                cout<<"The List is empty, nothing to display"<<endl;
371.                return;
372.            }
373.            s = start;
374.            for (i = 0;i < counter-1;i++)
375.            {
376.                cout<<s->info<<"<->";
377.                s = s->next;
378.            }
379.            cout<<s->info<<endl;
380.        }
381.        /*
382.         * Reverse Doubly Circular Linked List
383.         */
384.        void double_clist::reverse()
385.        {
386.            if (start == last && start == NULL)
387.            {
388.                cout<<"The List is empty, nothing to reverse"<<endl;
389.                return;
390.            }
391.            struct node *p1, *p2;
392.            p1 = start;
393.            p2 = p1->next;
394.            p1->next = NULL;
395.            p1->prev = p2;
396.            while (p2 != start)
397.            {
398.                p2->prev = p2->next;
399.                p2->next = p1;
400.                p1 = p2;
401.                p2 = p2->prev;
402.            }
403.            last = start;
404.            start = p1;
405.            cout<<"List Reversed"<<endl;
406.        }
```

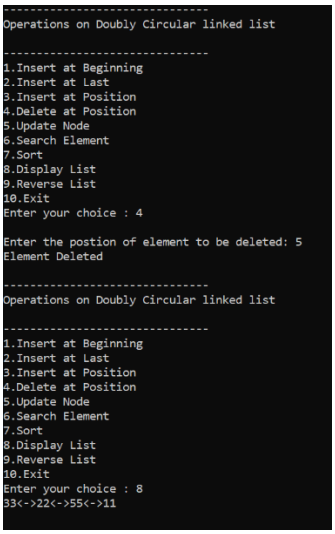
QUESTION

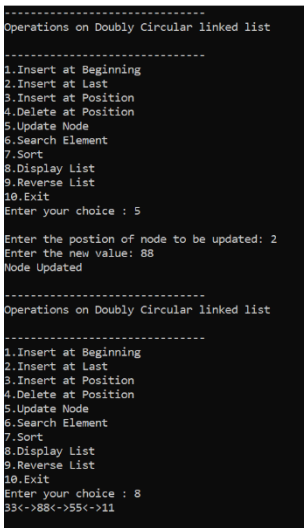
1. Insert 5 values then attach the output for operation 1 to 9.

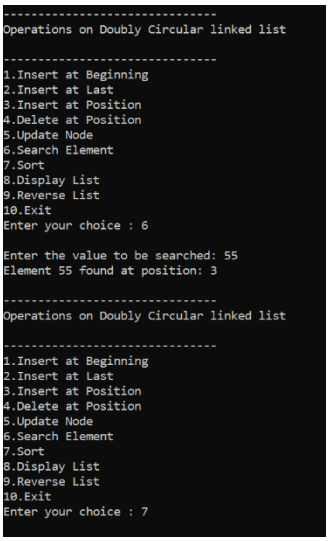


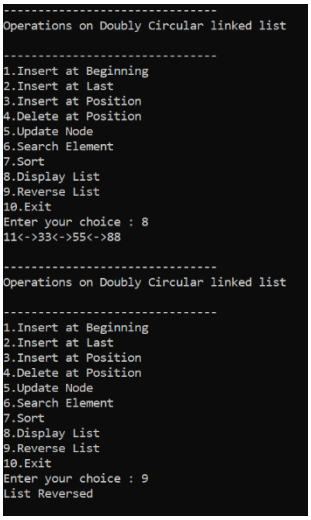


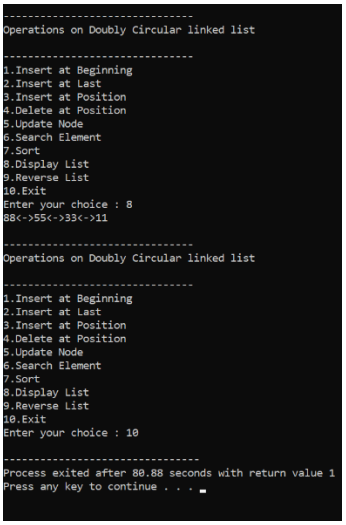












Question 2 (Circular Linked List)

```
#include<bits/stdc++.h>
using namespace std;

struct Node
{
    int data;                                                    //data of
the node
    struct Node *next;                                           //pointer
to the next node in the list
};

int Length(struct Node *head)                                    //function to
calculate the length of the linked list
{
    struct Node *t;
    int i = 0;
    if (head == NULL)
            //handle underflow condition
    {
        return 0;
    }

    t = head -> next;

    do

    {
                                    //handle traversal through the list
        t = t -> next;
        i++;
    } while (t != head->next);
    return i;
}

struct Node *Start(struct Node *head, int data)                  //function to insert
nodes at the beginning of the list
{
    struct Node *temp = (struct Node *) malloc(sizeof(struct Node));
    if (head == NULL)
                                    //handle underflow condition
        {
    temp -> data = data;
    head = temp;
    head -> next = head;
        }
    else
    {
```

```
    temp -> data = data;
    temp -> next = head -> next;
    head -> next = temp;
    head = temp;
    }
    return head;
}

struct Node *End(struct Node *head, int data)
            //function to insert nodes at the end of the list
{
    struct Node *temp = (struct Node *) malloc(sizeof(struct Node)), *a = head;
    if (head == NULL)
                                    //handle underflow condition
       {
    temp -> data = data;
    head = temp;
    head -> next = head;
       }
    else
    {
    do
    {
       a = a -> next;
    } while (a -> next != head);
            //traverse to the end of the list
    temp -> data = data;
    temp -> next = a -> next;
    a -> next = temp;
    }
    return head;
}

struct Node *Middle(struct Node *head, int data, int index)
            //function to insert nodes anywhere in between the beginning and the end
{
    if (head == NULL)
            //handle underflow condition
    {
       cout << "List is empty!" << endl;
       return NULL;
    }

    int len = Length(head);                                          //get the length of
the list for making a decision
    if (index > len || index < 0)                                    //wrong index
given
    {
       cout << "Wrong input, insertion not possible!" << endl;
```

```
        return head;
    }
    if (index == 0)                                                    //insert
at the beginning
    {
        head = Start(head,data);
        return head;
    }
    if (index == len)                                                  //insert
at the end
    {
        head = End(head,data);
        return head;
    }
    struct Node *temp = (struct Node *)malloc(sizeof(struct Node)), *a = head;
    do

    {
                                          //handle data traversal from beginning to end
        a = a -> next;
    } while (a -> next != head);
    len = 0;
    while (1)
    {
        if (len == index)                                              //handle
node addition
        {
            temp -> data = data;
            temp -> next = a -> next;
            a -> next = temp;
            return head;
        }
        a = a -> next;
        len++;
    }
}

struct Node *First(struct Node* head)
                                  //function to delete node from the start of the list
{
    struct Node *prev = head, *first = head;

    if (head == NULL) {
            //handle underflow condition
        cout << "List is empty" << endl;
        return NULL;
    }
```

```
    if (prev->next == prev)
                        //handle deletion for only one node of the list
    {
        head = NULL;
        return head;
    }

    while (prev->next != head)
    {
        prev = prev->next;
    }

    prev->next = first->next;

    head = prev->next;
    free(first);
    return head;
}

struct Node *Last(struct Node* head)
                        //function to delete node from the end of the list
{
    struct Node *curr = head, *temp = head, *prev;

    if (head == NULL) {
                                //handle underflow condition
        cout << "List is empty" << endl;
        return NULL;
    }

    if (curr->next == curr)
                        //handle deletion for only one node of the list
    {
        head = NULL;
        return head;
    }

    while (curr->next != head)
    {
        prev = curr;
        curr = curr->next;
    }

    prev->next = curr->next;
    head = prev->next;
    free(curr);
    return head;
}
```

```
struct Node *Position(struct Node* head, int index)
            //function to delete data from anywhere in between the start and the end of the list
{
   int len = Length(head);
   int count = 1;
   struct Node *prev = head, *next = head;

   if (head == NULL) {
                                    //handle underflow condition
      cout << "List is empty" << endl;
      return NULL;
   }

   if (index > len || index < 0)
            //wrong index entered
   {
      cout << "Wrong data given, deletion not possible!" << endl;
      return head;
   }

   if (index == 0)
   {
      First(head);
      return head;
   }

   if (index == len)
   {
      Last(head);
      return head;
   }

   while (len > 0)
   {
                                          //traverse through the list and delete the node in the list
      if (index == count)
      {
         prev->next = next->next;
         free(next);
         return head;
      }
      prev = prev->next;
      next = prev->next;
      len--;
      count++;
   }

   return head;
}
```

```
void Display(struct Node *head)
                                        //function to traverse throughout the list
{
   struct Node *t;
   if (head == NULL)
   {
      cout << "Linked list is empty." << endl;
      return;
   }

   t = head;

   do
   {
      cout << t -> data << " -> ";
      t = t -> next;
   } while (t != head);
   cout<<" "<<endl;
}

int main()
{
   struct Node *head = NULL;
   int n = 0,a = 0;
   char ch;
   while (1)
   {
   cout << " 1. Add at the beginning\n 2. Add at the end \n 3. Add at index \n 4. Display list \n 5.
Remove from the start \n 6. Remove from the end \n 7. Remove from index \n 5. Quit" << endl;
   cin >> ch;
   switch (ch)
   {
      case '1' : cout << "Enter data" << endl;
                 cin >> n;
                 head = Start(head, n);
                 break;
      case '2' : cout << "Enter data" << endl;
                 cin >> n;
                 head = End(head, n);
                 break;
      case '3' : cout << "Enter data" << endl;
                 cin >> n;
                 cout << "Enter index to insert at" << endl;
                 cin >> a;
                 head = Middle(head, n, a);
                 break;
      case '4' : Display(head);
                 break;
```

```
        case '5' :  head = First(head);
                    break;
        case '6' :  head = Last(head);
                    break;
        case '7' :  cout << "Enter index to insert at" << endl;
                    cin >> a;
                    head  = Position(head,a);
                    break;
        case '8' : exit(0);
        default : cout << "Wrong choice!" << endl;
    }
    }
    return 0;
}
```

QUESTION

1. Insert 10 values then attach the output.