C++ Program to Implement Stack using array
****

```
#include <iostream>
using namespace std;
int stack[100], n=100, top=-1;
void push(int val) {
   if(top>=n-1)
   cout<<"Stack Overflow"<<endl;
   else {
      top++;
      stack[top]=val;
   }
}
void pop() {
   if(top<=-1)
   cout<<"Stack Underflow"<<endl;
   else {
      cout<<"The popped element is "<< stack[top] <<endl;
      top--;
   }
}
void display() {
   if(top>=0) {
      cout<<"Stack elements are:";
      for(int i=top; i>=0; i--)
      cout<<stack[i]<<" ";
      cout<<endl;
   } else
   cout<<"Stack is empty";
}
int main() {
   int ch, val;
   cout<<"1) Push in stack"<<endl;
   cout<<"2) Pop from stack"<<endl;
   cout<<"3) Display stack"<<endl;
   cout<<"4) Exit"<<endl;
   do {
      cout<<"Enter choice: "<<endl;
      cin>>ch;
      switch(ch) {
         case 1: {
            cout<<"Enter value to be pushed:"<<endl;
            cin>>val;
```

```
            push(val);
            break;
        }
        case 2: {
            pop();
            break;
        }
        case 3: {
            display();
            break;
        }
        case 4: {
            cout<<"Exit"<<endl;
            break;
        }
        default: {
            cout<<"Invalid Choice"<<endl;
        }
      }
   }while(ch!=4);
   return 0;
}
```

C++ Program to Implement Queue using Array

```
*****************************************

#include <iostream>
using namespace std;
int queue[100], n = 100, front = - 1, rear = - 1;
void Insert() {
   int val;
   if (rear == n - 1)
   cout<<"Queue Overflow"<<endl;
   else {
      if (front == - 1)
      front = 0;
      cout<<"Insert the element in queue : "<<endl;
      cin>>val;
      rear++;
      queue[rear] = val;
   }
}
void Delete() {
   if (front == - 1 || front > rear) {
```

```
    cout<<"Queue Underflow ";
    return ;
  } else {
    cout<<"Element deleted from queue is : "<< queue[front] <<endl;
    front++;;
  }
}
void Display() {
  if (front == - 1)
  cout<<"Queue is empty"<<endl;
  else {
    cout<<"Queue elements are : ";
    for (int i = front; i <= rear; i++)
    cout<<queue[i]<<" ";
      cout<<endl;
  }
}
int main() {
  int ch;
  cout<<"1) Insert element to queue"<<endl;
  cout<<"2) Delete element from queue"<<endl;
  cout<<"3) Display all the elements of queue"<<endl;
  cout<<"4) Exit"<<endl;
  do {
    cout<<"Enter your choice : "<<endl;
    cin<<ch;
    switch (ch) {
      case 1: Insert();
      break;
      case 2: Delete();
      break;
      case 3: Display();
      break;
      case 4: cout<<"Exit"<<endl;
      break;
      default: cout<<"Invalid choice"<<endl;
    }
  } while(ch!=4);
  return 0;
}
```

C++ Program to Implement Dequeue

****

```
#include<iostream>
using namespace std;
#define SIZE 10
class dequeue {
  int a[20],f,r;
  public:
    dequeue();
    void insert_at_beg(int);
    void insert_at_end(int);
    void delete_fr_front();
    void delete_fr_rear();
    void show();
};
dequeue::dequeue() {
  f=-1;
  r=-1;
}
void dequeue::insert_at_end(int i) {
  if(r>=SIZE-1) {
    cout<<"\n insertion is not possible, overflow!!!!";
  } else {
    if(f==-1) {
      f++;
      r++;
    } else {
      r=r+1;
    }
    a[r]=i;
    cout<<"\nInserted item is"<<a[r];
  }
}
void dequeue::insert_at_beg(int i) {
  if(f==-1) {
    f=0;
    a[++r]=i;
    cout<<"\n inserted element is:"<<i;
  } else if(f!=0) {
    a[--f]=i;
    cout<<"\n inserted element is:"<<i;
  } else {
    cout<<"\n insertion is not possible, overflow!!!";
  }
}
void dequeue::delete_fr_front() {
```

```
if(f==-1) {
  cout<<"deletion is not possible::dequeue is empty";
  return;
}
else {
  cout<<"the deleted element is:"<<a[f];
  if(f==r) {
    f=r=-1;
    return;
  } else
    f=f+1;
  }
}
void dequeue::delete_fr_rear() {
  if(f==-1) {
    cout<<"deletion is not possible::dequeue is empty";
    return;
  }
  else {
    cout<<"the deleted element is:"<<a[r];
    if(f==r) {
      f=r=-1;
    } else
      r=r-1;
  }
}
void dequeue::show() {
  if(f==-1) {
    cout<<"Dequeue is empty";
  } else {
    for(int i=f;i<=r;i++) {
      cout<<a[i]<<" ";
    }
  }
}
int main() {
  int c,i;
  dequeue d;
  Do//perform switch opeartion {
  cout<<"\n 1.insert at beginning";
  cout<<"\n 2.insert at end";
  cout<<"\n 3.show";
  cout<<"\n 4.deletion from front";
  cout<<"\n 5.deletion from rear";
```

```
      cout<<"\n 6.exit";
      cout<<"\n enter your choice:";
      cin>>c;
      switch(c) {
        case 1:
          cout<<"enter the element to be inserted";
          cin>>i;
          d.insert_at_beg(i);
        break;
        case 2:
          cout<<"enter the element to be inserted";
          cin>>i;
          d.insert_at_end(i);
        break;
        case 3:
          d.show();
        break;
        case 4:
          d.delete_fr_front();
        break;
        case 5:
          d.delete_fr_rear();
        break;
        case 6:
          exit(1);
        break;
        default:
          cout<<"invalid choice";
        break;
      }
   } while(c!=7);
}
```

C++ Program to Implement Singly Linked List

```
****************************************

#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};
```

```
struct Node* head = NULL;
void insert(int new_data) {
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   new_node->data = new_data;
   new_node->next = head;
   head = new_node;
}
void display() {
   struct Node* ptr;
   ptr = head;
   while (ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
   }
}
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<"The linked list is: ";
   display();
   return 0;
}
```

C++ Program to Implement Doubly Linked List

```
**********************************************

#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *prev;
   struct Node *next;
};
struct Node* head = NULL;
void insert(int newdata) {
   struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
   newnode->data = newdata;
   newnode->prev = NULL;
   newnode->next = head;
   if(head != NULL)
```

```
   head->prev = newnode ;
   head = newnode;
}
void display() {
   struct Node* ptr;
   ptr = head;
   while(ptr != NULL) {
      cout<< ptr->data <<" ";
      ptr = ptr->next;
   }
}
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<"The doubly linked list is: ";
   display();
   return 0;
}
```

11111111111111**********1111111
****

```
// A complete working C program to demonstrate all
// insertion before a given node
#include <stdio.h>
#include <stdlib.h>

// A linked list node
struct Node {
        int data;
        struct Node* next;
        struct Node* prev;
};

/* Given a reference (pointer to pointer) to the head of a list
and an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
        struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
```

```
	new_node->data = new_data;

	new_node->next = (*head_ref);
	new_node->prev = NULL;

	if ((*head_ref) != NULL)
		(*head_ref)->prev = new_node;

	(*head_ref) = new_node;
}

/* Given a node as next_node, insert a new node before the given node */
void insertBefore(struct Node** head_ref, struct Node* next_node, int new_data)
{
	/*1. check if the given next_node is NULL */
	if (next_node == NULL) {
		printf("the given next node cannot be NULL");
		return;
	}

	/* 2. allocate new node */
	struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

	/* 3. put in the data */
	new_node->data = new_data;

	/* 4. Make prev of new node as prev of next_node */
	new_node->prev = next_node->prev;

	/* 5. Make the prev of next_node as new_node */
	next_node->prev = new_node;

	/* 6. Make next_node as next of new_node */
	new_node->next = next_node;

	/* 7. Change next of new_node's previous node */
	if (new_node->prev != NULL)
		new_node->prev->next = new_node;
	/* 8. If the prev of new_node is NULL, it will be
	the new head node */
	else
		(*head_ref) = new_node;

}
```

```
// This function prints contents of linked list starting from the given node
void printList(struct Node* node)
{
	struct Node* last;
	printf("\nTraversal in forward direction \n");
	while (node != NULL) {
		printf(" %d ", node->data);
		last = node;
		node = node->next;
	}

	printf("\nTraversal in reverse direction \n");
	while (last != NULL) {
		printf(" %d ", last->data);
		last = last->prev;
	}
}

/* Driver program to test above functions*/
int main()
{
	/* Start with the empty list */
	struct Node* head = NULL;
	push(&head, 7);

	push(&head, 1);

	push(&head, 4);

	// Insert 8, before 1. So linked list becomes 4->8->1->7->NULL
	insertBefore(&head, head->next, 8);

	printf("Created DLL is: ");
	printList(head);

	getchar();
	return 0;
}
```

C++ Program to Implement Circular Singly Linked List
****

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};
struct Node* head = NULL;
void insert(int newdata) {
   struct Node *newnode = (struct Node *)malloc(sizeof(struct Node));
   struct Node *ptr = head;
   newnode->data = newdata;
   newnode->next = head;
   if (head!= NULL) {
      while (ptr->next != head)
      ptr = ptr->next;
      ptr->next = newnode;
   } else
   newnode->next = newnode;
   head = newnode;
}
void display() {
   struct Node* ptr;
   ptr = head;
   do {
      cout<<ptr->data <<" ";
      ptr = ptr->next;
   } while(ptr != head);
}
int main() {
   insert(3);
   insert(1);
   insert(7);
   insert(2);
   insert(9);
   cout<<"The circular linked list is: ";
   display();
   return 0;
}
```