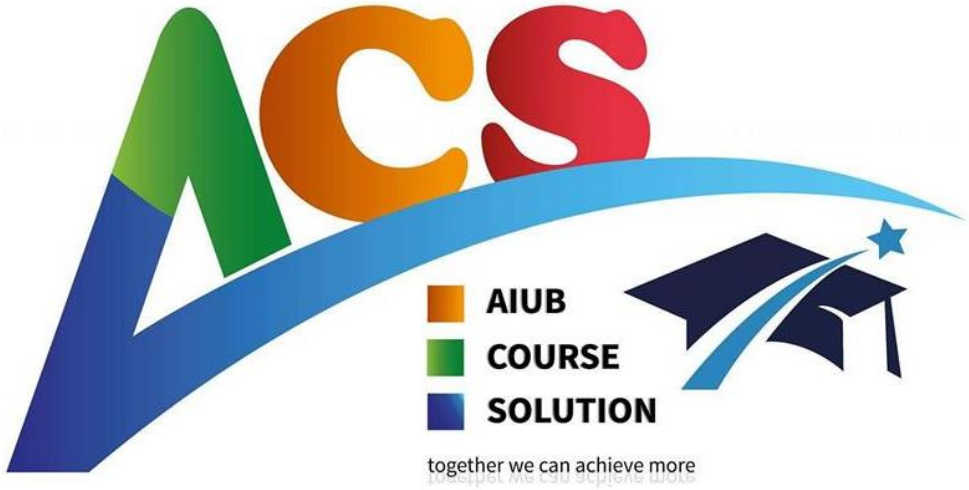

COURSE: DATA STRUCTURE

CHAPTER: *STACK,QUEUE,LINKED LIST ALL CODES FOR MID TERM*

PART-01

SOLVED BY:

MAHMUD HASAN (PRACCHAD)

AIUB COURSE SOLUTION-ACS

LINK = https://www.youtube.com/channel/UCC3KjA8kstFtM-2CxVr-jcg

AIUB COURSE SOLUTION

LINK= https://www.facebook.com/groups/aiubcoursesolution/

STACK

Stacks Using Arrays

```
#include < iostream.h >
#include < stdlib.h >
#include < stdio.h >

class IntStack
{
public:
   IntStack(int num) {  top = 0; maxelem = num;
s = new int[maxelem]; }
   void  push(int t)
   {
      if (top == maxelem) return;
      s[top++] = t;
   }
   int pop()
   {
      if (top == 0) return -1;
      return s[--top];
   }
   void display()
   {
     if (top == 0) { cout << "(empty)\n";
return; }
     for (int t=0 ; t < top ; t++) cout << s[t]
<< " ";
     cout << "\n";
   }
   int   empty()  {  return top == 0;  }
private:
```

```
    int *s;
    int top;
    int maxelem;
};

void main()
{
    IntStack *s = new IntStack(100);
    int d;

    s->display();
    s->push(1);
    s->display();
    s->push(2);
    s->display();
    s->push(3);
    s->display();
    s->push(4);
    s->display();
    s->pop();
    s->display();
    s->pop();
    s->display();
    s->push(10);
    s->display();
    s->pop();
    s->display();
    s->pop();
    s->display();
    s->pop();
    s->display();
    s->pop();
    s->display();
    s->pop();
    s->display();
}
```

Stack Implementation using Array in C++

```
#include<iostream>
#define MAX 5

using namespace std;


int STACK[MAX],TOP;

//stack initialization
void initStack(){
   TOP=-1;
}
//check it is empty or not
int isEmpty(){
   if(TOP==-1)
      return 1;
   else
      return 0;
}

//check stack is full or not
int isFull(){
   if(TOP==MAX-1)
      return 1;
   else
      return 0;
}
void push(int num){
   if(isFull()){
      cout<<"STACK is FULL."<<endl;
      return;
```

```
    }
    ++TOP;
    STACK[TOP]=num;
    cout<<num<<" has been inserted."<<endl;
}

void display(){
    int i;
    if(isEmpty()){
        cout<<"STACK is EMPTY."<<endl;
        return;
    }
    for(i=TOP;i>=0;i--){
        cout<<STACK[i]<<" ";
    }
    cout<<endl;
}
//pop - to remove item
void pop(){
    int temp;
    if(isEmpty()){
        cout<<"STACK is EMPTY."<<endl;
        return;
    }

    temp=STACK[TOP];
    TOP--;
    cout<<temp<<" has been deleted."<<endl;
}
int main(){
    int num;
    initStack();
    char ch;
    do{
```

```
        int a;
        cout<<"Chosse \n1.push\n"<<"2.pop\n"<<"3.display\n";
        cout<<"Please enter your choice: ";
        cin>>a;
        switch(a)
        {
          case 1:
            cout<<"Enter an Integer Number: ";
            cin>>num;
            push(num);
          break;

case 2:
            pop();
            break;

          case 3:
            display();
            break;

          default :
          cout<<"An Invalid Choice!!!\n";



        }
        cout<<"Do you want to continue ? ";
        cin>>ch;
        }while(ch=='Y'||ch=='y');
    return 0;
}
```

Stack Code

```
#include<iostream>
using namespace std;
#define limit 10

int stack[limit], top=0;

void push(int value)
{
   if(top<limit)
   {
      stack[top]=value;
      top++;
   }
   else
   {
      cout<<"Stack Over Flowed\n";
   }
}

void pop()
{
   if(top<0)
   {
```

```
        cout<<"Stack is empty\n";
    }
    else
    {
        top--;
        cout<<stack[top]<<" ";
    }
}

int main()
{
    push(10);
    push(20);
    push(30);
    pop();
    pop();
    pop();
    pop();
    pop();
    return 0;
}
```

QUEUE

Queue - Circular Array implementation

```
#include<iostream>
using namespace std;
#define MAX_SIZE 101  //maximum size of the array that will store Queue.

// Creating a class named Queue.
class Queue
{
private:
        int A[MAX_SIZE];
        int front, rear;
public:
        // Constructor - set front and rear as -1.
        // We are assuming that for an empty Queue, both front and rear will be -1.
        Queue()
        {
         front = -1;
         rear = -1;
        }

        // To check wheter Queue is empty or not
        bool IsEmpty()
```

```
{
 return (front == -1 && rear == -1);
}

// To check whether Queue is full or not
bool IsFull()
{
 return (rear+1)%MAX_SIZE == front ? true : false;
}

// Inserts an element in queue at rear end
void Enqueue(int x)
{
 cout<<"Enqueuing "<<x<<" \n";
 if(IsFull())
 {
        cout<<"Error: Queue is Full\n";
        return;
 }
 if (IsEmpty())
 {
        front = rear = 0;
 }
 else
 {
    rear = (rear+1)%MAX_SIZE;
```

```
 }
 A[rear] = x;
}

// Removes an element in Queue from front end.
void Dequeue()
{
 cout<<"Dequeuing \n";
 if(IsEmpty())
 {
        cout<<"Error: Queue is Empty\n";
        return;
 }
 else if(front == rear )
 {
        rear = front = -1;
 }
 else
 {
        front = (front+1)%MAX_SIZE;
 }
}
// Returns element at front of queue.
int Front()
{
 if(front == -1)
```

```
        {
                cout<<"Error: cannot return front from empty queue\n";
                return -1;
        }
        return A[front];
    }
    /*
      Printing the elements in queue from front to rear.
      This function is only to test the code.
      This is not a standard function for Queue implementation.
    */
    void Print()
    {
      // Finding number of elements in queue
      int count = (rear+MAX_SIZE-front)%MAX_SIZE + 1;
      cout<<"Queue      : ";
      for(int i = 0; i <count; i++)
      {
                int index = (front+i) % MAX_SIZE; // Index of element while
travesing circularly from front
                cout<<A[index]<<" ";
      }
      cout<<"\n\n";
    }
};
int main()
```

```
{
        /*Driver Code to test the implementation
          Printing the elements in Queue after each Enqueue or Dequeue
        */
  Queue Q; // creating an instance of Queue.
  Q.Enqueue(2);  Q.Print();
  Q.Enqueue(4);  Q.Print();
  Q.Enqueue(6);  Q.Print();
  Q.Dequeue();   Q.Print();
  Q.Enqueue(8);  Q.Print();
}
```

implement the Queue ADT using an array

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;

class queue
{
        int queue1[5];
        int rear,front;
    public:
        queue()
         {
            rear=-1;
            front=-1;
         }
        void insert(int x)
        {
           if(rear >  4)
            {
              cout <<"queue over flow";
              front=rear=-1;
              return;
            }
            queue1[++rear]=x;
```

```
            cout <<"inserted" <<x;
        }
        void delet()
        {
            if(front==rear)
            {
                cout <<"queue under flow";
                return;
            }
            cout <<"deleted" <<queue1[++front];
        }
        void display()
        {
            if(rear==front)
            {
                cout <<" queue empty";
                return;
            }
            for(int i=front+1;i<=rear;i++)
            cout <<queue1[i]<<" ";
        }
};

main()
{
    int ch;
```

```
    queue qu;
    while(1)
     {
        cout <<"\n1.insert  2.delet  3.display  4.exit\nEnter ur choice";
        cin >> ch;
        switch(ch)
         {
           case 1:    cout <<"enter the element";
                      cin >> ch;
                     qu.insert(ch);
                     break;
           case 2:  qu.delet();  break;
           case 3:  qu.display();break;
           case 4: exit(0);
           }
       }
return (0);
}
```

Linked List

The C++ code for the creation of new a node would like this:

```
void createnode(int value)
 {
   node *temp=new node;
   temp->data=value;
   temp->next=NULL;
   if(head==NULL)
   {
     head=temp;
     tail=temp;
     temp=NULL;
   }
   else
   {
     tail->next=temp;
     tail=temp;
   }
 }
```

The code for displaying nodes of linked list is given below:

```
void display()
{
  node *temp=new node;
  temp=head;
  while(temp!=NULL)
  {
    cout<<temp->data<<"\t";
    temp=temp->next;
  }
}
```

C++ code for insertion of node would be as follows:

```
void insert_position(int pos, int value)
{
  node *pre=new node;
  node *cur=new node;
  node *temp=new node;
```

```
    cur=head;
    for(int i=1;i<pos;i++)
    {
      pre=cur;
      cur=cur->next;
    }
    temp->data=value;
    pre->next=temp;
    temp->next=cur;
  }
```

Deletation :

```
void delete_last()
  {
    node *current=new node;
    node *previous=new node;
    current=head;
    while(current->next!=NULL)
    {
      previous=current;
      current=current->next;
```

```
  }
  tail=previous;
  previous->next=NULL;
  delete current;
}
```

The deletion can be done in C++ by using code given below:

```
void delete_position(int pos)
{
  node *current=new node;
  node *previous=new node;
  current=head;
  for(int i=1;i<pos;i++)
  {
    previous=current;
    current=current->next;
  }
  previous->next=current->next;
}
```

Linked list code:

```
#include <iostream>
#include <cstdlib>


class Node
{
public:
    Node* next;
    int data;
};

using namespace std;

class LinkedList
{
public:
    int length;
    Node* head;

    LinkedList();
```

```
    ~LinkedList();
    void add(int data);
    void print();
};

LinkedList::LinkedList(){
    this->length = 0;
    this->head = NULL;
}

LinkedList::~LinkedList(){
    std::cout << "LIST DELETED";
}

void LinkedList::add(int data){
    Node* node = new Node();
    node->data = data;
    node->next = this->head;
    this->head = node;
    this->length++;
}
```

```
void LinkedList::print(){
    Node* head = this->head;
    int i = 1;
    while(head){
        std::cout << i << ": " << head->data << std::endl;
        head = head->next;
        i++;
    }
}

int main(int argc, char const *argv[])
{
    LinkedList* list = new LinkedList();
    for (int i = 0; i < 100; ++i)
    {
        list->add(rand() % 100);
    }
    list->print();
    std::cout << "List Length: " << list->length << std::endl;
    delete list;
    return 0;
}
```

Another Code for Linked List(Full Form):

```
*
 * C++ Program to Implement Singly Linked List
 */
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;
/*
 * Node Declaration
 */
struct node
{
    int info;
    struct node *next;
}*start;

/*
 * Class Declaration
 */
class single_llist
{
```

```
    public:
        node* create_node(int);
        void insert_begin();
        void insert_pos();
        void insert_last();
        void delete_pos();
        void sort();
        void search();
        void update();
        void reverse();
        void display();
        single_llist()
        {
            start = NULL;
        }
};

/*
 * Main :contains menu
 */
main()
{
```

```
int choice, nodes, element, position, i;
single_llist sl;
start = NULL;
while (1)
{
    cout<<endl<<"--------------------------------"<<endl;
    cout<<endl<<"Operations on singly linked list"<<endl;
    cout<<endl<<"--------------------------------"<<endl;
    cout<<"1.Insert Node at beginning"<<endl;
    cout<<"2.Insert node at last"<<endl;
    cout<<"3.Insert node at position"<<endl;
    cout<<"4.Sort Link List"<<endl;
    cout<<"5.Delete a Particular Node"<<endl;
    cout<<"6.Update Node Value"<<endl;
    cout<<"7.Search Element"<<endl;
    cout<<"8.Display Linked List"<<endl;
    cout<<"9.Reverse Linked List "<<endl;
    cout<<"10.Exit "<<endl;
    cout<<"Enter your choice : ";
    cin>>choice;
    switch(choice)
    {
```

```
case 1:
    cout<<"Inserting Node at Beginning: "<<endl;
    sl.insert_begin();
    cout<<endl;
    break;
case 2:
    cout<<"Inserting Node at Last: "<<endl;
    sl.insert_last();
    cout<<endl;
    break;
case 3:
    cout<<"Inserting Node at a given position:"<<endl;
    sl.insert_pos();
    cout<<endl;
    break;
case 4:
    cout<<"Sort Link List: "<<endl;
    sl.sort();
    cout<<endl;
    break;
case 5:
    cout<<"Delete a particular node: "<<endl;
```

```
        sl.delete_pos();
        break;
    case 6:
        cout<<"Update Node Value:"<<endl;
        sl.update();
        cout<<endl;
        break;
    case 7:
        cout<<"Search element in Link List: "<<endl;
        sl.search();
        cout<<endl;
        break;
    case 8:
        cout<<"Display elements of link list"<<endl;
        sl.display();
        cout<<endl;
        break;
    case 9:
        cout<<"Reverse elements of Link List"<<endl;
        sl.reverse();
        cout<<endl;
        break;
```

```
        case 10:
            cout<<"Exiting..."<<endl;
            exit(1);
            break;
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
}

/*
 * Creating Node
 */
node *single_llist::create_node(int value)
{
    struct node *temp, *s;
    temp = new(struct node);
    if (temp == NULL)
    {
        cout<<"Memory not allocated "<<endl;
        return 0;
    }
```

```
    else
    {
        temp->info = value;
        temp->next = NULL;
        return temp;
    }
}

/*
 * Inserting element in beginning
 */
void single_llist::insert_begin()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *p;
    temp = create_node(value);
    if (start == NULL)
    {
        start = temp;
        start->next = NULL;
```

```
    }
    else
    {
        p = start;
        start = temp;
        start->next = p;
    }
    cout<<"Element Inserted at beginning"<<endl;
}

/*
 * Inserting Node at last
 */
void single_llist::insert_last()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *s;
    temp = create_node(value);
    s = start;
    while (s->next != NULL)
```

```
    {
        s = s->next;
    }
    temp->next = NULL;
    s->next = temp;
    cout<<"Element Inserted at last"<<endl;
}

/*
 * Insertion of node at a given position
 */
void single_llist::insert_pos()
{
    int value, pos, counter = 0;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *s, *ptr;
    temp = create_node(value);
    cout<<"Enter the postion at which node to be inserted: ";
    cin>>pos;
    int i;
    s = start;
```

```
while (s != NULL)
{
    s = s->next;
    counter++;
}
if (pos == 1)
{
    if (start == NULL)
    {
        start = temp;
        start->next = NULL;
    }
    else
    {
        ptr = start;
        start = temp;
        start->next = ptr;
    }
}
else if (pos > 1  && pos <= counter)
{
    s = start;
```

```
        for (i = 1; i < pos; i++)
        {
            ptr = s;
            s = s->next;
        }
        ptr->next = temp;
        temp->next = s;
    }
    else
    {
        cout<<"Positon out of range"<<endl;
    }
}

/*
 * Sorting Link List
 */
void single_llist::sort()
{
    struct node *ptr, *s;
    int value;
    if (start == NULL)
```

```
    {
        cout<<"The List is empty"<<endl;
        return;
    }
    ptr = start;
    while (ptr != NULL)
    {
        for (s = ptr->next;s !=NULL;s = s->next)
        {
            if (ptr->info > s->info)
            {
                value = ptr->info;
                ptr->info = s->info;
                s->info = value;
            }
        }
        ptr = ptr->next;
    }
}


/*
 * Delete element at a given position
```

```
 */
void single_llist::delete_pos()
{
    int pos, i, counter = 0;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the position of value to be deleted: ";
    cin>>pos;
    struct node *s, *ptr;
    s = start;
    if (pos == 1)
    {
        start = s->next;
    }
    else
    {
        while (s != NULL)
        {
            s = s->next;
```

```
        counter++;
    }
    if (pos > 0 && pos <= counter)
    {
        s = start;
        for (i = 1;i < pos;i++)
        {
            ptr = s;
            s = s->next;
        }
        ptr->next = s->next;
    }
    else
    {
        cout<<"Position out of range"<<endl;
    }
    free(s);
    cout<<"Element Deleted"<<endl;
  }
}

/*
```

```
 * Update a given Node
 */
void single_llist::update()
{
    int value, pos, i;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the node postion to be updated: ";
    cin>>pos;
    cout<<"Enter the new value: ";
    cin>>value;
    struct node *s, *ptr;
    s = start;
    if (pos == 1)
    {
        start->info = value;
    }
    else
    {
```

```
        for (i = 0;i < pos - 1;i++)
        {
            if (s == NULL)
            {
                cout<<"There are less than "<<pos<<" elements";
                return;
            }
            s = s->next;
        }
        s->info = value;
    }
    cout<<"Node Updated"<<endl;
}

/*
 * Searching an element
 */
void single_llist::search()
{
    int value, pos = 0;
    bool flag = false;
    if (start == NULL)
```

```
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the value to be searched: ";
    cin>>value;
    struct node *s;
    s = start;
    while (s != NULL)
    {
        pos++;
        if (s->info == value)
        {
            flag = true;
            cout<<"Element "<<value<<" is found at position
"<<pos<<endl;
        }
        s = s->next;
    }
    if (!flag)
        cout<<"Element "<<value<<" not found in the list"<<endl;
}
```

```
/*
 * Reverse Link List
 */
void single_llist::reverse()
{
    struct node *ptr1, *ptr2, *ptr3;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    if (start->next == NULL)
    {
        return;
    }
    ptr1 = start;
    ptr2 = ptr1->next;
    ptr3 = ptr2->next;
    ptr1->next = NULL;
    ptr2->next = ptr1;
    while (ptr3 != NULL)
```

```
    {
        ptr1 = ptr2;
        ptr2 = ptr3;
        ptr3 = ptr3->next;
        ptr2->next = ptr1;
    }
    start = ptr2;
}

/*
 * Display Elements of a link list
 */
void single_llist::display()
{
    struct node *temp;
    if (start == NULL)
    {
        cout<<"The List is Empty"<<endl;
        return;
    }
    temp = start;
    cout<<"Elements of list are: "<<endl;
```

```
    while (temp != NULL)
    {
        cout<<temp->info<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;
}
```

Student id,cgpa,age in linked list:

```
#include<iostream>
using namespace std;

struct student
{
   int studentId;
   float studentCGPA;
   int studentAge;
   student *next;
};

student *list, *nptr, *tptr;

student * createNode(int sid, float scgpa, int sage)
{
   student *ptr = new student;
   ptr->studentId = sid;
   ptr->studentCGPA = scgpa;
   ptr->studentAge = sage;
   return ptr;
}
```

```
void createLink(student *ptr)
{
   if(list==NULL)
   {
      list = ptr;
      tptr= ptr;
   }
   else{
      tptr->next = ptr;
      tptr= ptr;
   }
}

void displayList()
{
   tptr = list;

   while(tptr!=NULL)
   {
      cout<<"ID: "<<tptr->studentId<<" CGPA: "<<tptr-
>studentCGPA<<" "<<tptr->studentAge<<endl;
```

```
        tptr = tptr->next;
    }
}

student *beginning()
{
    return list;
}

student *end()
{
    tptr= list;
    while(tptr->next!=NULL)
        tptr = tptr->next;
    //cout<<"Student id of last node is: "<<tptr->studentId<<endl;
    return tptr;
}

void insertBack(student *ptr)
{
    nptr = createNode(5,4,17);
    ptr->next = nptr;
```

```
    ptr= ptr->next;
}

int main()
{
    list= NULL;

    nptr = createNode(1, 3.24, 18);
    createLink(nptr);
    nptr = createNode(2, 3.14, 18);
    createLink(nptr);
    nptr = createNode(3, 3.34, 18);
    createLink(nptr);
    nptr = createNode(4, 3.44, 18);
    createLink(nptr);
    displayList();
    tptr = end();
    insertBack(tptr);
    displayList();
    end();
}
```