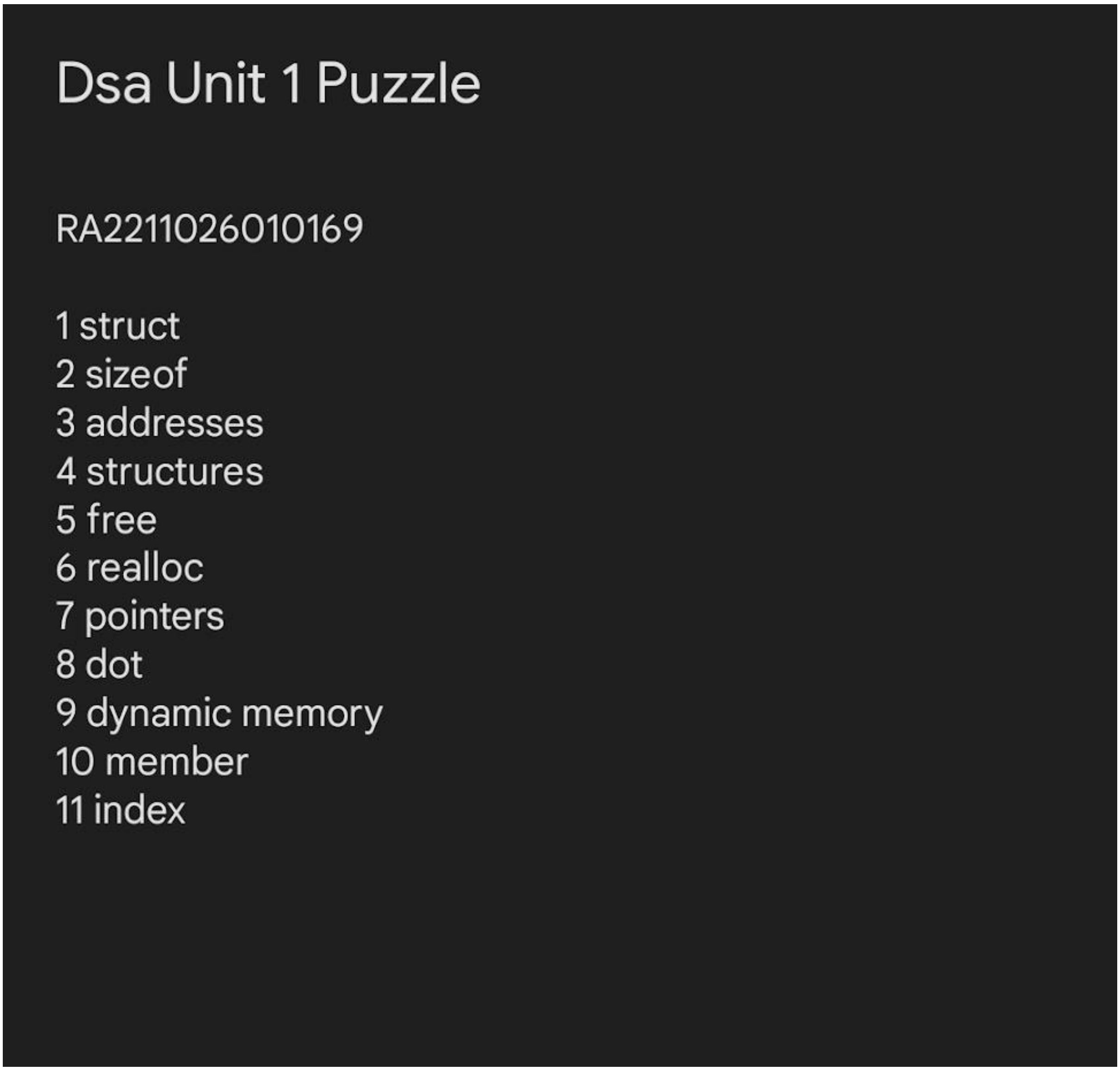
YATHARTH PATIDAR

RA2211026010169

CSE – AIML & Y1

DSA ASSIGMENTS

1.UNIT 1 PUZZLE



Dsa Unit 1 Puzzle

RA2211026010169

1 struct
2 sizeof
3 addresses
4 structures
5 free
6 realloc
7 pointers
8 dot
9 dynamic memory
10 member
11 index

2. Group activity_operations on arrays

```
#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;

//insert function

void insert(int *arr1 ,int size1,int ele){
```

```
int *newarr1 = (int*)malloc(sizeof(int)*size1+1);
for(int i=0;i&lt;size1;i++){
newarr1[i]=arr1[i];
}
newarr1[size1]=ele;
free(arr1);
for(int i=0;i&lt;size1+1;i++){
printf(&quot;%d&quot;,newarr1[i]);
}
printf(&quot;\n&quot;);
}
//delete function
void delete(int *arr2 ,int size2,int pos){
int *newarr2=(int*)malloc(sizeof(int)*size2-1);
for(int i=0;i&lt;pos;i++){

newarr2[i]=arr2[i];


}
for(int i=pos+1;i&lt;size2;i++){

newarr2[i-1]=arr2[i];


}
free(arr2);
for(int i=0;i&lt;size2-1;i++){
printf(&quot;%d&quot;,newarr2[i]);


}
printf(&quot;\n&quot;);
}
```

```
//search function
void search(int *arr3, int size3, int elem){
for(int i=0;i&lt;size3;i++){
if(arr3[i]==elem){
printf(&quot;Found at %d&quot;,i);
break;
}
}
printf(&quot;\n&quot;);
}
//sort function
void sort(int *arr4,int size4){
int a;
for (int i = 0; i &lt; size4; ++i)
{
for (int j = i + 1; j &lt; size4; ++j){
if (arr4[i] &gt; arr4[j]) {
a = arr4[i];
arr4[i] = arr4[j];
arr4[j] = a;

}
}
}
for(int i=0;i&lt;size4;i++){
printf(&quot;%d&quot;,arr4[i]);
}
printf(&quot;\n&quot;);
}

int main()
```

```
{
int *arr1;
int *arr2;
int size=4;
arr1=(int*)malloc(sizeof(int)*size);
for(int i=0;i&lt;size;i++){
scanf(&quot;%d&quot;,&amp;arr1[i]);
}
arr2=(int*)malloc(sizeof(int)*size);
for(int i=0;i&lt;size;i++){
arr2[i]=arr1[i];
}
insert(arr1,size,5);
delete(arr2,size,2);
search(arr1,size,2);
sort(arr1,size);

return 0;
}
```

3. Singly Linked List

```
#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;

// Define the structure for a singly linked list node
struct Node {
int data;
struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
```

```
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
printf(&quot;Memory allocation failed.\n&quot;);
exit(1); }
newNode-&gt;data = value;
newNode-&gt;next = NULL;
return newNode;
}
// Function to insert a node at the beginning
struct Node* insertAtBeginning(struct Node* head, int value) {
struct Node* newNode = createNode(value);
newNode-&gt;next = head;
return newNode;
}
// Function to insert a node at the end
void insertAtEnd(struct Node* head, int value) {
struct Node* newNode = createNode(value);
struct Node* current = head;

while (current-&gt;next != NULL) {

current = current-&gt;next; }
current-&gt;next = newNode; }

// Function to insert a node at any position
void insertAtPosition(struct Node* head, int value, int position) {
struct Node* newNode = createNode(value);
struct Node* current = head;

for (int i = 1; i &lt; position - 1; i++) {
current = current-&gt;next;
if (current == NULL) {
printf(&quot;Invalid position.\n&quot;);
return;
}
}

newNode-&gt;next = current-&gt;next;
current-&gt;next = newNode;
}

// Function to delete a node at the beginning
struct Node* deleteAtBeginning(struct Node* head) {
if (head == NULL) {
printf(&quot;List is empty. Cannot delete.\n&quot;);
return NULL;
}
struct Node* temp = head;
head = head-&gt;next;
free(temp);
return head;
}

// Function to delete a node at the end
void deleteAtEnd(struct Node* head) {
if (head == NULL) {
printf(&quot;List is empty. Cannot delete.\n&quot;);
return;
}
struct Node* current = head;
struct Node* prev = NULL;
```

```
while (current-&gt;next != NULL) {
prev = current;
current = current-&gt;next;
}

prev-&gt;next = NULL;
free(current);
}

// Function to delete a node at any position
void deleteAtPosition(struct Node* head, int position) {
if (head == NULL) {
printf(&quot;List is empty. Cannot delete.\n&quot;);
return;
}

struct Node* current = head;
struct Node* prev = NULL;

for (int i = 1; i &lt; position; i++) {
prev = current;

current = current-&gt;next;
if (current == NULL) {
printf(&quot;Invalid position.\n&quot;);
return;
}
}

prev-&gt;next = current-&gt;next;
free(current);
```

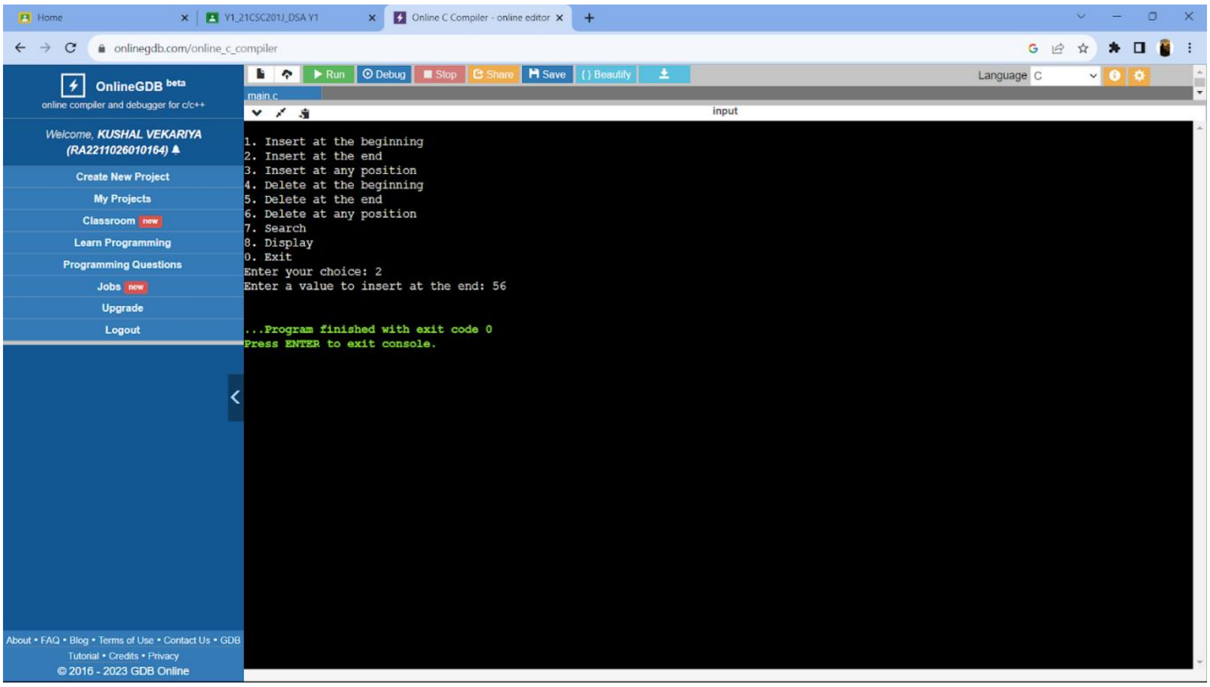
```
}

// Function to search for a value in the list
int search(struct Node* head, int value) {
struct Node* current = head;
int position = 1;

while (current != NULL) {
if (current-&gt;data == value) {
return position; }
current = current-&gt;next;
position++;
}

return -1; // Value not found }

// Function to display the linked list
void display(struct Node* head) {
struct Node* current = head;

while (current != NULL) {
printf(&quot;%d &quot;, current-&gt;data);

current = current-&gt;next; }
printf(&quot;NULL\n&quot;); }

int main() {
struct Node* head = NULL;
int choice, value, position;

do {
```

```
printf(&quot;\n1. Insert at the beginning\n&quot;);
printf(&quot;2. Insert at the end\n&quot;);
printf(&quot;3. Insert at any position\n&quot;);
printf(&quot;4. Delete at the beginning\n&quot;);
printf(&quot;5. Delete at the end\n&quot;);
printf(&quot;6. Delete at any position\n&quot;);
printf(&quot;7. Search\n&quot;);
printf(&quot;8. Display\n&quot;);
printf(&quot;0. Exit\n&quot;);
printf(&quot;Enter your choice: &quot;);
scanf(&quot;%d&quot;, &amp;choice);

switch (choice) {
case 1:
printf(&quot;Enter a value to insert at the beginning: &quot;);
scanf(&quot;%d&quot;, &amp;value);
head = insertAtBeginning(head, value);
break;

case 2:
printf(&quot;Enter a value to insert at the end: &quot;);
scanf(&quot;%d&quot;, &amp;value);
insertAtEnd(head, value);

break;

case 3:
printf(&quot;Enter a value to insert: &quot;);
scanf(&quot;%d&quot;, &amp;value);
printf(&quot;Enter the position to insert at: &quot;);
scanf(&quot;%d&quot;, &amp;position);
```

```
insertAtPosition(head, value, position);
break;

case 4:
head = deleteAtBeginning(head);
break;

case 5:
deleteAtEnd(head);
break;

case 6:
printf(&quot;Enter the position to delete: &quot;);
scanf(&quot;%d&quot;, &amp;position);
deleteAtPosition(head, position);
break;

case 7:
printf(&quot;Enter a value to search for: &quot;);
scanf(&quot;%d&quot;, &amp;value);
position = search(head, value);
if (position != -1) {
printf(&quot;Value found at position %d\n&quot;, position);
} else {

printf(&quot;Value not found in the list\n&quot;);
}
break;

case 8:
display(head);
```

```
break;


case 0:
printf(&quot;Exiting the program.\n&quot;);
break;
default:
printf(&quot;Invalid choice. Please try again.\n&quot;);
break;
}
} while (choice != 0);
return 0;
}
```

OUTPUT:



```
1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Delete at the beginning
5. Delete at the end
6. Delete at any position
7. Search
8. Display
0. Exit
Enter your choice: 2
Enter a value to insert at the end: 56


...Program finished with exit code 0
Press ENTER to exit console.
```

4. Singly Circular Linked List

```
#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;


// Define the structure for a singly circular linked list node
struct Node {
```

```
int data;
struct Node* next;
};

// Function to create a new node
struct Node* createNode(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
printf(&quot;Memory allocation failed.\n&quot;);
exit(1);
}
newNode-&gt;data = value;
newNode-&gt;next = NULL;
return newNode;
}

// Function to insert a node at the beginning
struct Node* insertAtBeginning(struct Node* head, int value) {
struct Node* newNode = createNode(value);
if (head == NULL) {
newNode-&gt;next = newNode; // Point to itself for circularity
return newNode;
}

struct Node* tail = head;
while (tail-&gt;next != head) {
tail = tail-&gt;next;
}

newNode-&gt;next = head;
tail-&gt;next = newNode;
```

```
return newNode;
}

// Function to insert a node at the end
struct Node* insertAtEnd(struct Node* head, int value) {
struct Node* newNode = createNode(value);
if (head == NULL) {
newNode-&gt;next = newNode; // Point to itself for circularity
return newNode;
}

struct Node* tail = head;
while (tail-&gt;next != head) {
tail = tail-&gt;next;
}

newNode-&gt;next = head;
tail-&gt;next = newNode;
return head;
}

// Function to insert a node at any position
struct Node* insertAtPosition(struct Node* head, int value, int position) {
struct Node* newNode = createNode(value);

if (head == NULL) {
newNode-&gt;next = newNode; // Point to itself for circularity
return newNode;
}

if (position == 1) {
```

```
newNode-&gt;next = head;
struct Node* tail = head;
while (tail-&gt;next != head) {
tail = tail-&gt;next;
}
tail-&gt;next = newNode;
return newNode;
}

struct Node* current = head;
int i;
for (i = 1; i &lt; position - 1; i++) {
current = current-&gt;next;
if (current == head) {
printf(&quot;Invalid position.\n&quot;);
return head;
}
}

newNode-&gt;next = current-&gt;next;
current-&gt;next = newNode;
return head;
}

// Function to delete a node at the beginning

struct Node* deleteAtBeginning(struct Node* head) {
if (head == NULL) {
printf(&quot;List is empty. Cannot delete.\n&quot;);
return NULL;
}
```

```
struct Node* tail = head;
while (tail-&gt;next != head) {
tail = tail-&gt;next;
}

if (head == tail) {
free(head);
return NULL;
}

struct Node* temp = head;
head = head-&gt;next;
tail-&gt;next = head;
free(temp);
return head;
}

// Function to delete a node at the end
struct Node* deleteAtEnd(struct Node* head) {
if (head == NULL) {
printf(&quot;List is empty. Cannot delete.\n&quot;);
return NULL;
}

struct Node* tail = head;

struct Node* prev = NULL;

while (tail-&gt;next != head) {
prev = tail;
```

```
tail = tail-&gt;next;
}

if (head == tail) {
free(head);
return NULL;
}

prev-&gt;next = head;
free(tail);
return head;
}

// Function to delete a node at any position
struct Node* deleteAtPosition(struct Node* head, int position) {
if (head == NULL) {
printf(&quot;List is empty. Cannot delete.\n&quot;);
return NULL;
}

if (position == 1) {
struct Node* tail = head;
while (tail-&gt;next != head) {
tail = tail-&gt;next;
}

if (head == tail) {


free(head);
return NULL;
}
```

```
struct Node* temp = head;
head = head-&gt;next;
tail-&gt;next = head;
free(temp);
return head;
}

struct Node* current = head;
struct Node* prev = NULL;
int i;

for (i = 1; i &lt; position; i++) {
prev = current;
current = current-&gt;next;
if (current == head) {
printf(&quot;Invalid position.\n&quot;);
return head;
}
}

prev-&gt;next = current-&gt;next;
free(current);
return head;
}

// Function to search for a value in the list
int search(struct Node* head, int value) {

if (head == NULL) {
printf(&quot;List is empty. Cannot search.\n&quot;);
```

```
return -1;
}

struct Node* current = head;
int position = 1;

do {
if (current-&gt;data == value) {
return position;
}
current = current-&gt;next;
position++;
} while (current != head);

return -1; // Value not found
}

// Function to display the linked list
void display(struct Node* head) {
if (head == NULL) {
printf(&quot;List is empty.\n&quot;);
return;
}

struct Node* current = head;

do {
printf(&quot;%d &quot;, current-&gt;data);
current = current-&gt;next;

} while (current != head);
```

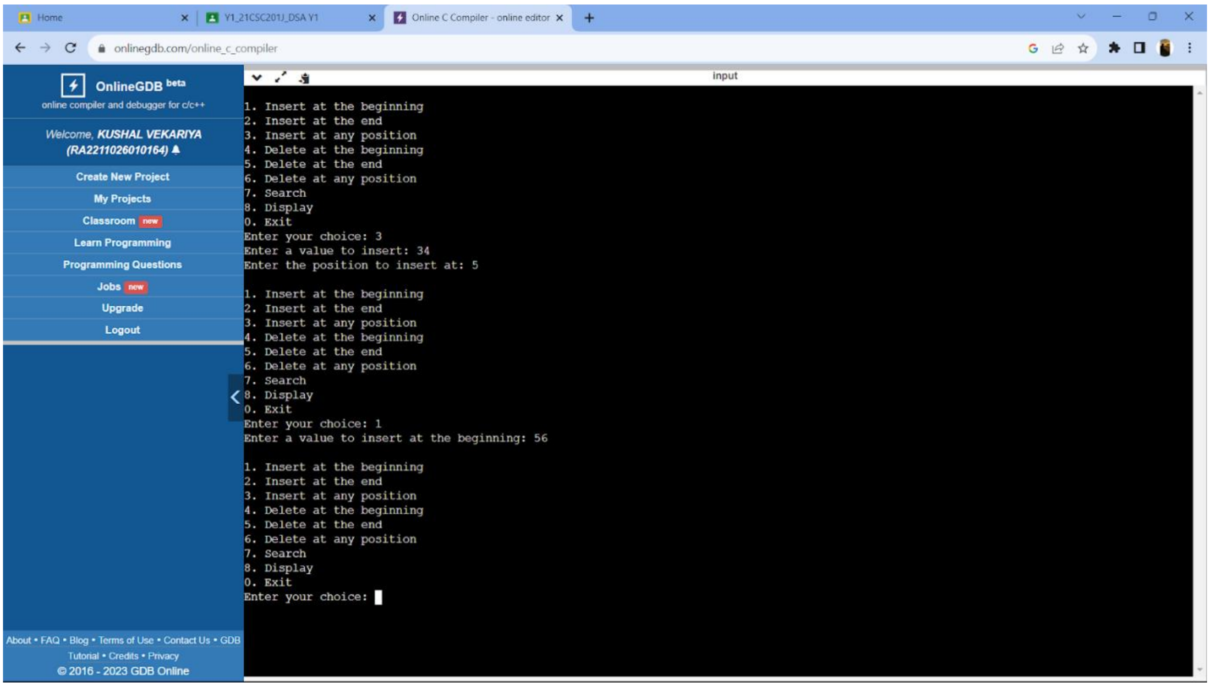
```
printf(&quot;%d \n&quot;, current-&gt;data); // Circular reference, use &#39;...&#39; to indicate
it&#39;s circular
}

int main() {
struct Node* head = NULL;
int choice, value, position;

do {
printf(&quot;\n1. Insert at the beginning\n&quot;);
printf(&quot;2. Insert at the end\n&quot;);
printf(&quot;3. Insert at any position\n&quot;);
printf(&quot;4. Delete at the beginning\n&quot;);
printf(&quot;5. Delete at the end\n&quot;);
printf(&quot;6. Delete at any position\n&quot;);
printf(&quot;7. Search\n&quot;);
printf(&quot;8. Display\n&quot;);
printf(&quot;0. Exit\n&quot;);
printf(&quot;Enter your choice: &quot;);
scanf(&quot;%d&quot;, &amp;choice);

switch (choice) {
case 1:
printf(&quot;Enter a value to insert at the beginning: &quot;);
scanf(&quot;%d&quot;, &amp;value);
head = insertAtBeginning(head, value);
break;

case 2:
printf(&quot;Enter a value to insert at the end: &quot;);
```

```
scanf(&quot;%d&quot;, &amp;value);
head = insertAtEnd(head, value);
break;

case 3:
printf(&quot;Enter a value to insert: &quot;);
scanf(&quot;%d&quot;, &amp;value);
printf(&quot;Enter the position to insert at: &quot;);
scanf(&quot;%d&quot;, &amp;position);
head = insertAtPosition(head, value, position);
break;

case 4:
head = deleteAtBeginning(head);
break;

case 5:
head = deleteAtEnd(head);
break;

case 6:
printf(&quot;Enter the position to delete: &quot;);
scanf(&quot;%d&quot;, &amp;position);
head = deleteAtPosition(head, position);
break;

case 7:
printf(&quot;Enter a value to search for: &quot;);
scanf(&quot;%d&quot;, &amp;value);
position = search(head, value);
```

```
if (position != -1) {

printf(&quot;Value found at position %d\n&quot;, position);
} else {
printf(&quot;Value not found in the list\n&quot;);
}
break;

case 8:
display(head);
break;

case 0:
printf(&quot;Exiting the program.\n&quot;);
break;

default:
printf(&quot;Invalid choice. Please try again.\n&quot;);
break;
}
} while (choice != 0);

// Free memory before exiting
struct Node* current = head;
while (current != NULL) {
struct Node* temp = current;
current = current-&gt;next;
free(temp);
}

return 0;
```

```
}
```

OUTPUT:



```
1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Delete at the beginning
5. Delete at the end
6. Delete at any position
7. Search
8. Display
0. Exit
Enter your choice: 3
Enter a value to insert: 34
Enter the position to insert at: 5

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Delete at the beginning
5. Delete at the end
6. Delete at any position
7. Search
8. Display
0. Exit
Enter your choice: 1
Enter a value to insert at the beginning: 56

1. Insert at the beginning
2. Insert at the end
3. Insert at any position
4. Delete at the beginning
5. Delete at the end
6. Delete at any position
7. Search
8. Display
0. Exit
Enter your choice: 
```

5. Doubly Linked List

```
#include &lt;stdio.h&gt;

#include &lt;stdlib.h&gt;


// Define the structure for a doubly circular linked list node

struct Node {

int data;

struct Node *prev;

struct Node *next;

};


// Function to insert a node at the beginning of the list

void insertAtBeginning(struct Node **head, int value) {

struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));

newNode-&gt;data = value;


if (*head == NULL) {
```

```
newNode-&gt;prev = newNode;
newNode-&gt;next = newNode;
} else {
newNode-&gt;prev = (*head)-&gt;prev;
newNode-&gt;next = *head;
(*head)-&gt;prev-&gt;next = newNode;
(*head)-&gt;prev = newNode;
}

*head = newNode;
}

// Function to insert a node at the end of the list

void insertAtEnd(struct Node **head, int value) {
struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode-&gt;data = value;

if (*head == NULL) {
newNode-&gt;prev = newNode;
newNode-&gt;next = newNode;
*head = newNode;
} else {
newNode-&gt;prev = (*head)-&gt;prev;
newNode-&gt;next = *head;
(*head)-&gt;prev-&gt;next = newNode;
(*head)-&gt;prev = newNode;
}
}

// Function to insert a node at a specified position
```

```
void insertAtPosition(struct Node **head, int value, int position) {
if (position &lt;= 0) {
printf(&quot;Invalid position.\n&quot;);
return;
}

if (*head == NULL || position == 1) {
insertAtBeginning(head, value);
return;
}

struct Node *current = *head;
for (int i = 1; i &lt; position - 1 &amp;&amp; current-&gt;next != *head; ++i) {
current = current-&gt;next;


}

if (current-&gt;next == *head &amp;&amp; position != 1) {
printf(&quot;Position out of bounds.\n&quot;);
return;
}

struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode-&gt;data = value;
newNode-&gt;prev = current;
newNode-&gt;next = current-&gt;next;
current-&gt;next-&gt;prev = newNode;
current-&gt;next = newNode;
}

// Function to delete the first node
```

```
void deleteAtBeginning(struct Node **head) {
if (*head == NULL) {
printf(&quot;List is empty.\n&quot;);
return;
}

struct Node *toDelete = *head;

if ((*head)-&gt;next == *head) {
*head = NULL;
} else {
(*head)-&gt;prev-&gt;next = (*head)-&gt;next;
(*head)-&gt;next-&gt;prev = (*head)-&gt;prev;
*head = (*head)-&gt;next;
}

free(toDelete);
}

// Function to delete the last node
void deleteAtEnd(struct Node **head) {
if (*head == NULL) {
printf(&quot;List is empty.\n&quot;);
return;
}

struct Node *toDelete = (*head)-&gt;prev;

if ((*head)-&gt;next == *head) {
*head = NULL;
} else {
```

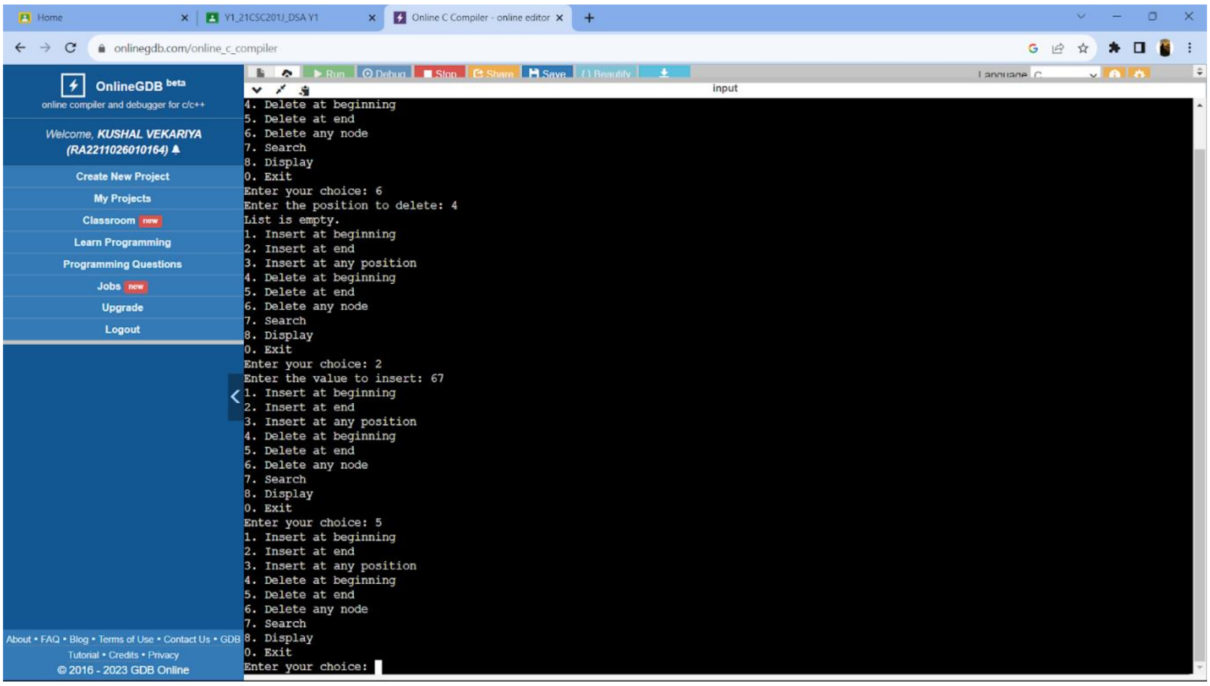
```
(*head)-&gt;prev = toDelete-&gt;prev;
toDelete-&gt;prev-&gt;next = *head;
}

free(toDelete);
}

// Function to delete a node at a specified position
void deleteAtPosition(struct Node **head, int position) {
if (*head == NULL) {
printf(&quot;List is empty.\n&quot;);
return;
}

if (position &lt;= 0) {

printf(&quot;Invalid position.\n&quot;);
return;
}

if (position == 1) {
deleteAtBeginning(head);
return;
}

struct Node *current = *head;
for (int i = 1; i &lt; position &amp;&amp; current-&gt;next != *head; ++i) {
current = current-&gt;next;
}

if (current-&gt;next == *head &amp;&amp; position != 1) {
```

```
printf(&quot;Position out of bounds.\n&quot;);
return;
}

current-&gt;prev-&gt;next = current-&gt;next;
current-&gt;next-&gt;prev = current-&gt;prev;
free(current);
}

// Function to search for a value in the list
int search(struct Node *head, int value) {
if (head == NULL) {
printf(&quot;List is empty.\n&quot;);
return -1;
}

struct Node *current = head;
int position = 1;
do {
if (current-&gt;data == value) {
return position;
}
current = current-&gt;next;
position++;
} while (current != head);

printf(&quot;Value not found in the list.\n&quot;);
return -1;
}

// Function to display the list
```

```
void display(struct Node *head) {
if (head == NULL) {
printf(&quot;List is empty.\n&quot;);
return;
}

struct Node *current = head;
do {
printf(&quot;%d &quot;, current-&gt;data);
current = current-&gt;next;
} while (current != head);
printf(&quot;\n&quot;);
}

int main() {
struct Node *head = NULL;

int choice, value, position;

do {
printf(&quot;1. Insert at beginning\n&quot;);
printf(&quot;2. Insert at end\n&quot;);
printf(&quot;3. Insert at any position\n&quot;);
printf(&quot;4. Delete at beginning\n&quot;);
printf(&quot;5. Delete at end\n&quot;);
printf(&quot;6. Delete any node\n&quot;);
printf(&quot;7. Search\n&quot;);
printf(&quot;8. Display\n&quot;);
printf(&quot;0. Exit\n&quot;);
printf(&quot;Enter your choice: &quot;);
scanf(&quot;%d&quot;, &amp;choice);
```

```
switch (choice) {

case 1:

printf(&quot;Enter the value to insert: &quot;);

scanf(&quot;%d&quot;, &amp;value);

insertAtBeginning(&amp;head, value);

break;

case 2:

printf(&quot;Enter the value to insert: &quot;);

scanf(&quot;%d&quot;, &amp;value);

insertAtEnd(&amp;head, value);

break;

case 3:

printf(&quot;Enter the value to insert: &quot;);

scanf(&quot;%d&quot;, &amp;value);

printf(&quot;Enter the position to insert at: &quot;);


scanf(&quot;%d&quot;, &amp;position);

insertAtPosition(&amp;head, value, position);

break;

case 4:

deleteAtBeginning(&amp;head);

break;

case 5:

deleteAtEnd(&amp;head);

break;

case 6:

printf(&quot;Enter the position to delete: &quot;);

scanf(&quot;%d&quot;, &amp;position);

deleteAtPosition(&amp;head, position);

break;
case 7:
printf(&quot;Enter the value to search: &quot;);
scanf(&quot;%d&quot;, &amp;value);
position = search(head, value);
if (position != -1) {
printf(&quot;Value found at position %d.\n&quot;, position);
}
break;
case 8:
display(head);
break;
case 0:
printf(&quot;Exiting program.\n&quot;);
break;
default:
printf(&quot;Invalid choice. Please enter a valid option.\n&quot;);
}

} while (choice != 0);

// Free the memory of remaining nodes before exiting
struct Node *current = head;
while (current != NULL) {
struct Node *temp = current;
current = current-&gt;next;
free(temp);
}

return 0;
}
```

OUTPUT:


Home
Y1_21CSC201J_DSA Y1
Online C Compiler - online editor
onlinegdb.com/online_c_compiler
OnlineGDB beta
online compiler and debugger for c/c++
Welcome, KUSHAL VEKARIYA (RA2211026010164)
Create New Project
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Upgrade
Logout
About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online
Run
Debug
Stop
Share
Save
Beautify
Language C
input

```
4. Delete at beginning
5. Delete at end
6. Delete any node
7. Search
8. Display
0. Exit
Enter your choice: 6
Enter the position to delete: 4
List is empty.
1. Insert at beginning
2. Insert at end
3. Insert at any position
4. Delete at beginning
5. Delete at end
6. Delete any node
7. Search
8. Display
0. Exit
Enter your choice: 2
Enter the value to insert: 67
1. Insert at beginning
2. Insert at end
3. Insert at any position
4. Delete at beginning
5. Delete at end
6. Delete any node
7. Search
8. Display
0. Exit
Enter your choice: 5
1. Insert at beginning
2. Insert at end
3. Insert at any position
4. Delete at beginning
5. Delete at end
6. Delete any node
7. Search
8. Display
0. Exit
Enter your choice: 
```

DATA STRUCTURES AND ALGORITHMS
Unit-II-Assignment

Reg. No. : RA2211026010169 Name : Yatharth Patidar

Year : ~~20~~ II Branch: CSE AIML Section: Y1

I. Choose the best answer:

1. Which of the following is not true about linked lists?
   (a)It is a collection of linked nodes.
   (b)It helps in dynamic allocation of memory space.
   [✓ (c)]It allows direct access to any of the nodes.
   (d)It requires more memory space in comparison to an array.

2. Which node pointers should be updated if a new node B is to be inserted in the middle of A and C nodes of a singly linked list?
   (a) NEXT pointer of A and NEXT pointer of C (b)NEXT pointer of B and NEXT pointer of C (c)NEXT pointer of B
   [✓ (d)]NEXT pointer of A and NEXT pointer of B

3. A circular linked list contains four nodes "A, B, C, D". Which node pointers should be updated if a new node E is to be inserted at end of the list?
   [✓ (a)]NEXT pointer of D and NEXT pointer of E
   (b)NEXT pointer of E
   (c)NEXT pointer of E and NEXT pointer of A
   (d)NEXT pointer of E and START POINTER

4. Which node pointers should be updated if a new node B is to be inserted in the middle of A and C nodes of a doubly linked list?
   (a) NEXT pointer of A, PREVIOUS pointer of B, NEXT pointer of C, and PREVIOUS pointer of C [✓ (b)]NEXT pointer of A, PREVIOUS pointer of B, NEXT pointer of B, and PREVIOUS pointer of C (c)NEXT pointer of A, PREVIOUS pointer of A, NEXT pointer of B, and PREVIOUS pointer of C (d)None of the above

5. Which of the following statements is true about doubly linked list?
   (a)It allows list traversal only in forward direction.
   (b)It allows list traversal only in forward direction.
   [✓ (c)]It allows list traversal in both forward and backward direction. (d)It allows complete list traversal starting from any of the nodes.

6. Which of the following statements is true about circular linked list?

[✓ (a)It allows complete list traversal starting from any of the nodes.]

(b)It allows complete list traversal only if we begin from the FIRST node.

(c) Like singly and doubly linked lists, the NEXT part of the last node of a circular linked list contains a NULL pointer indicating end of the list.

(d)None of the above

7. You are required to create a linked list for storing integer elements. Which of the following linked list implementations will require maximum amount of memory space?

(a) Singly linked [✓ (b)Doubly] (b)
linked
(c)Circular
(d)All of the above will occupy same space in memory

8. Which of the following linked list types allows you to print the list elements in reverse order?

[✓ (a)Doubly]
(b)Singly
(c)Circular
(d)None of the above

9. Which of the following is the fastest and easiest sorting technique?

(a)Bubble
[✓ (b)Quick]
(c)Insertion
(d)Bucket

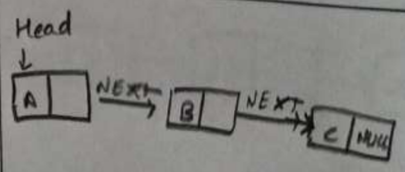
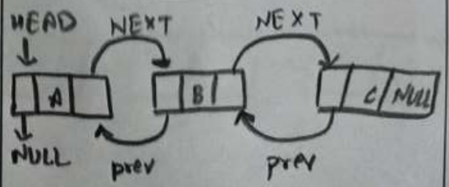
10. Which of the following searching techniques mandatorily requires the list to be already sorted?

(a) Li
near
[✓ (b)Binar
y]
(c)Hash
(d)None of the above

ANSWERS

1.	c	2.	d	3.	a	4.	b	5.	c	6.	a	7.	b	8.	a	9.	b	10.	10

II. Draw my node:

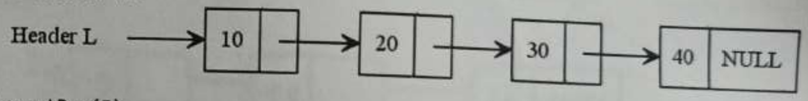
Singly Linked List	Doubly Linked List
	 HEAD NEXT NEXT A B C NULL NULL prev prev

III. Declare me:

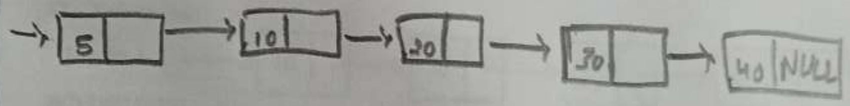
Singly Linked List	Doubly Linked List

struct node {
int data;
struct node *next;
}
	struct node {
	int data;
	struct node *next;
	struct node* prev;
	}

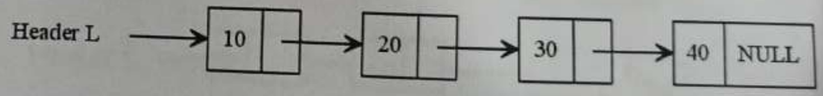
IV. Redraw me:



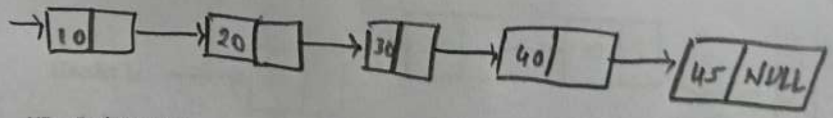
insertBeg(5)



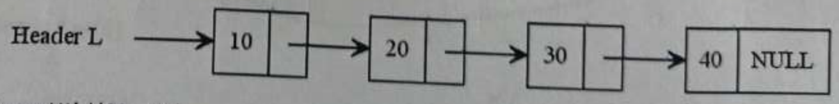
V. Redraw me:



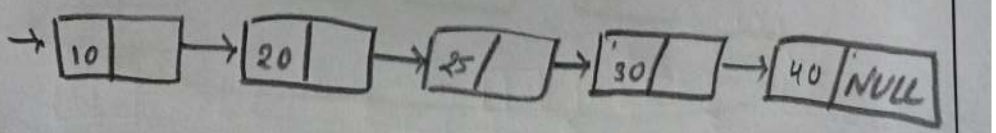
insertEnd(45)



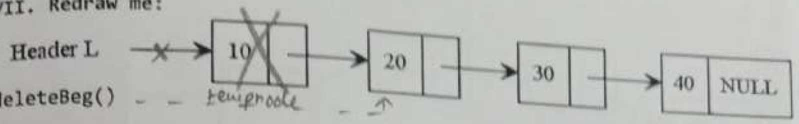
VI. Redraw me:



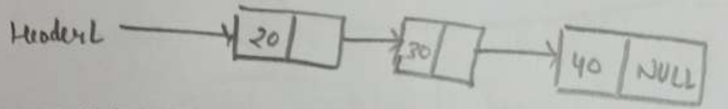
insertMid(20, 25)



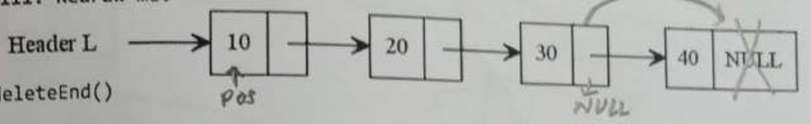
VII. Redraw me:



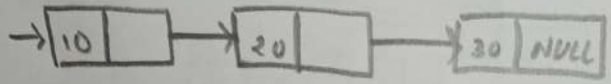
deleteBeg()



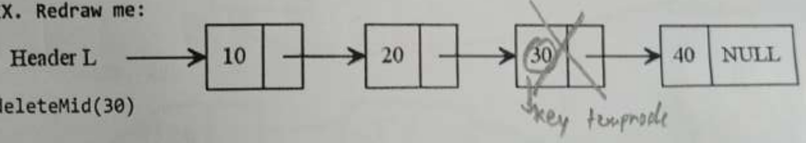
VIII. Redraw me:



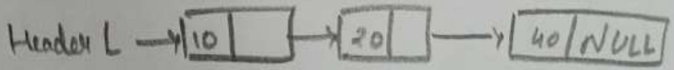
deleteEnd()



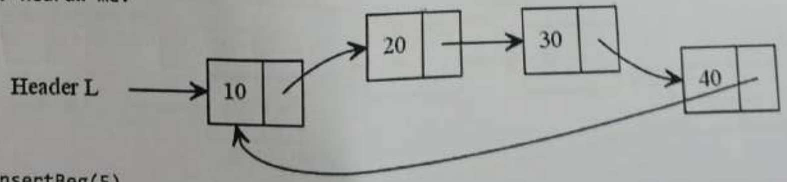
IX. Redraw me:



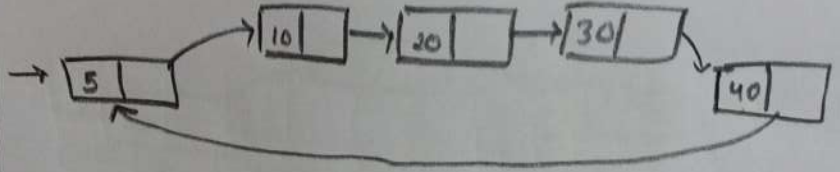
deleteMid(30)



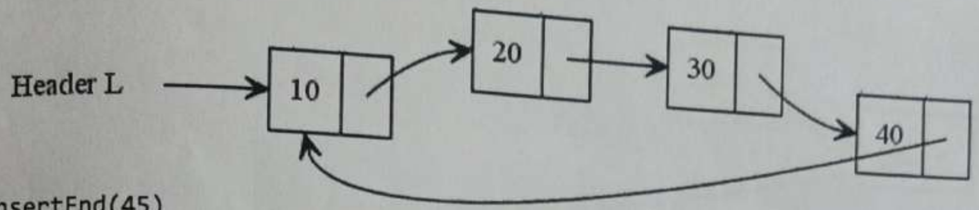
X. Redraw me:



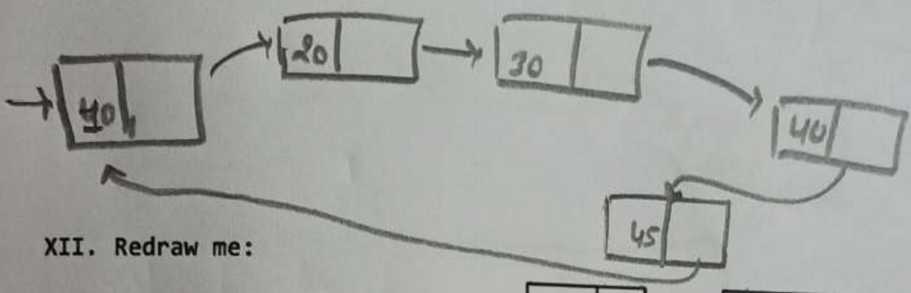
insertBeg(5)



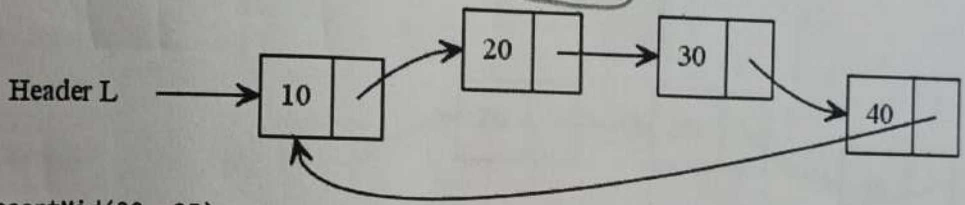
XI. Redraw me:



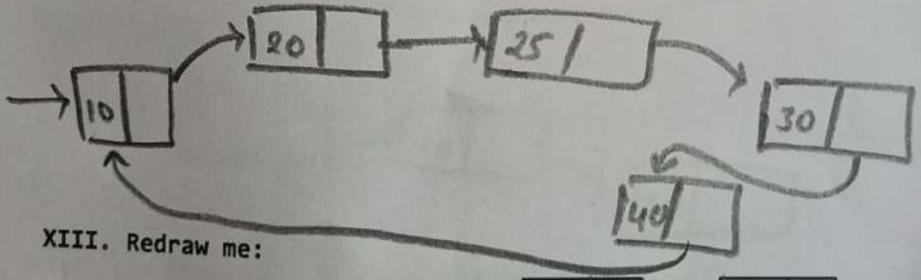
insertEnd(45)



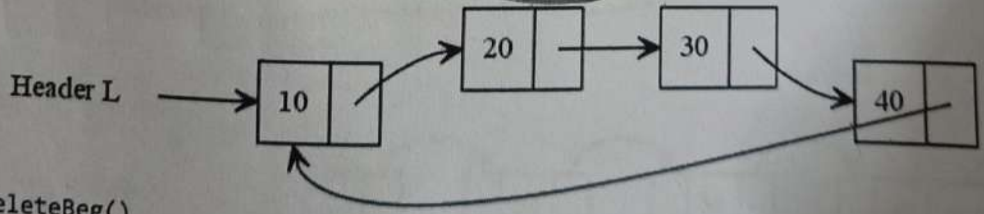
XII. Redraw me:



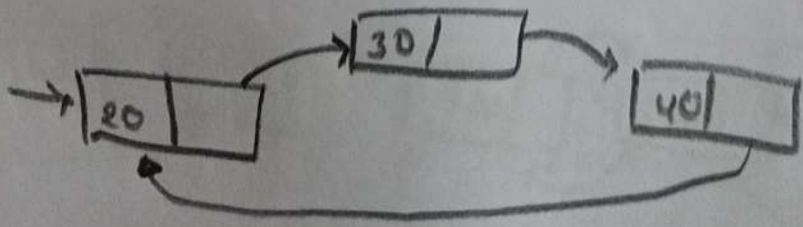
insertMid(20, 25)



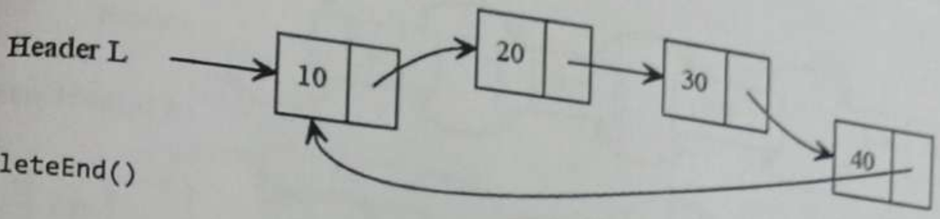
XIII. Redraw me:



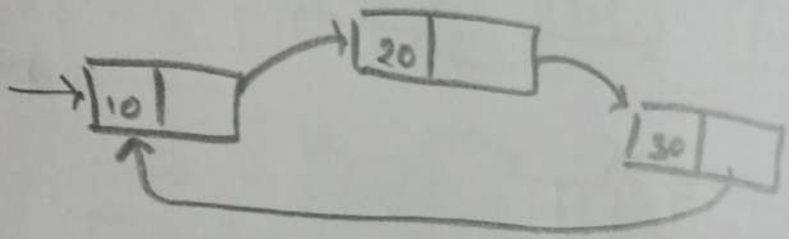
deleteBeg()



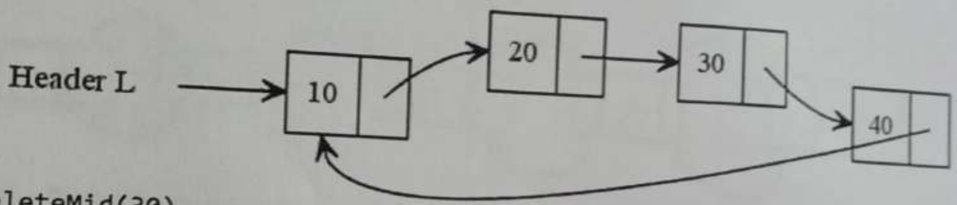
XIV. Redraw me:



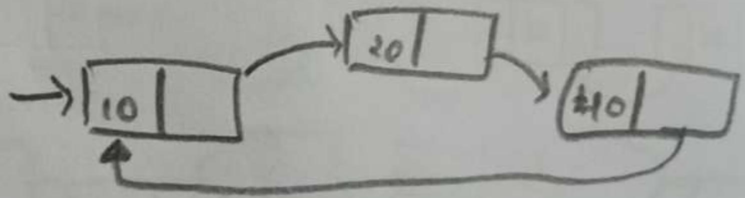
deleteEnd()



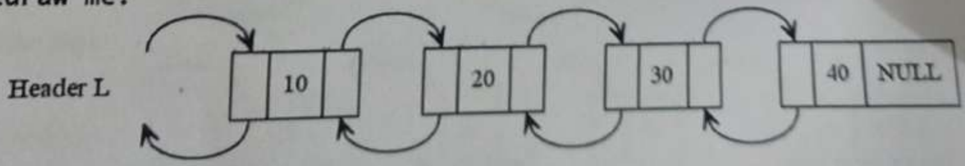
XV. Redraw me:



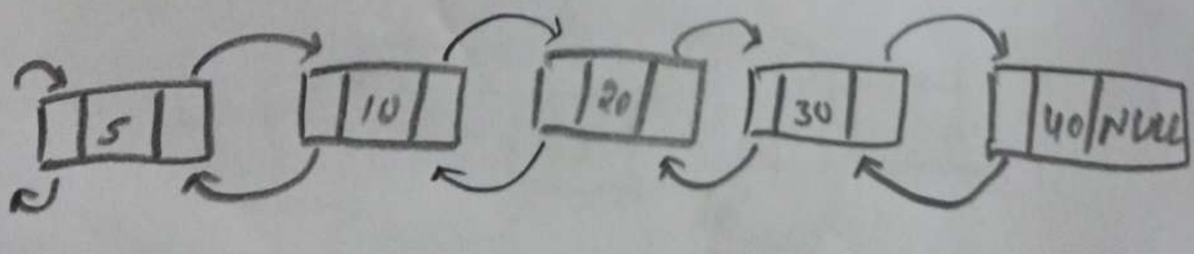
deleteMid(30)



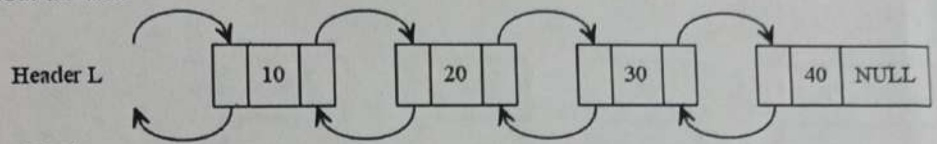
XVI. Redraw me:



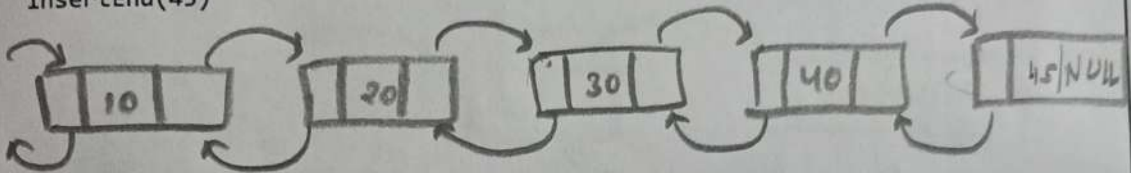
insertBeg(5)



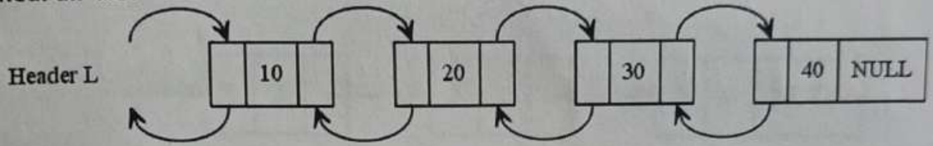
XVII. Redraw me:



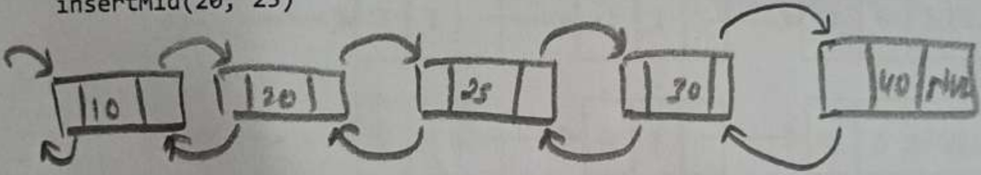
insertEnd(45)



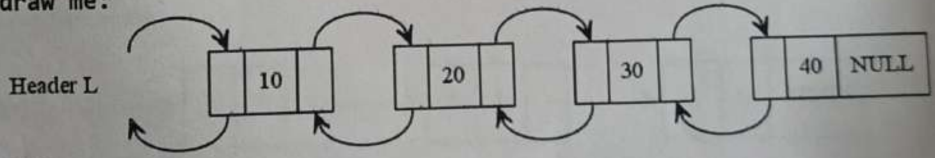
XVIII. Redraw me:



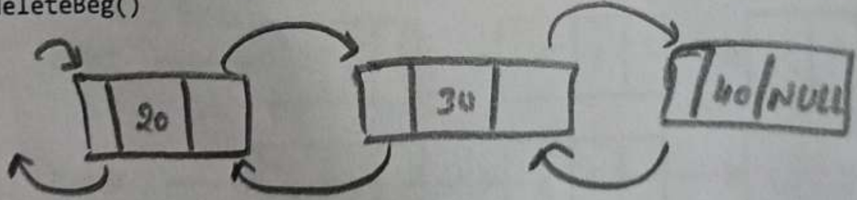
insertMid(20, 25)



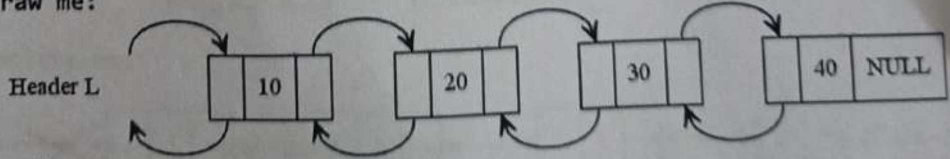
XIX. Redraw me:



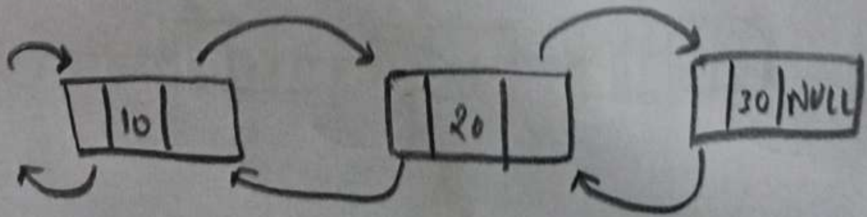
deleteBeg()



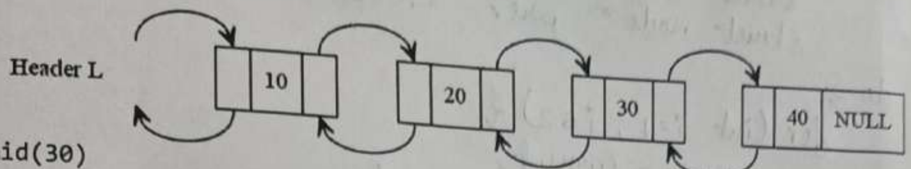
XX. Redraw me:



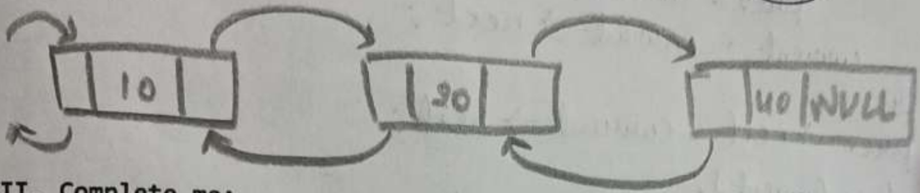
deleteEnd()



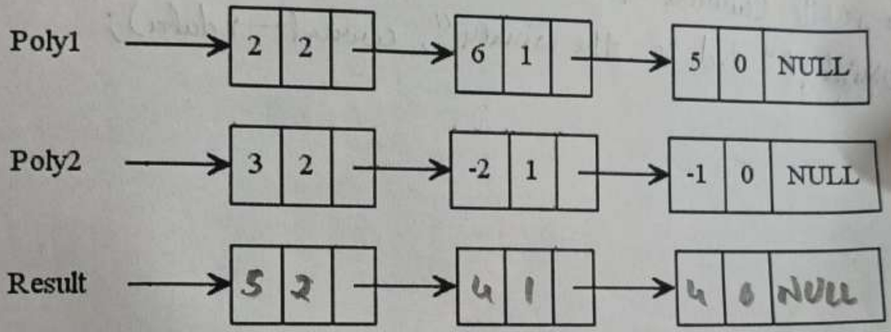
XXI. Redraw me:



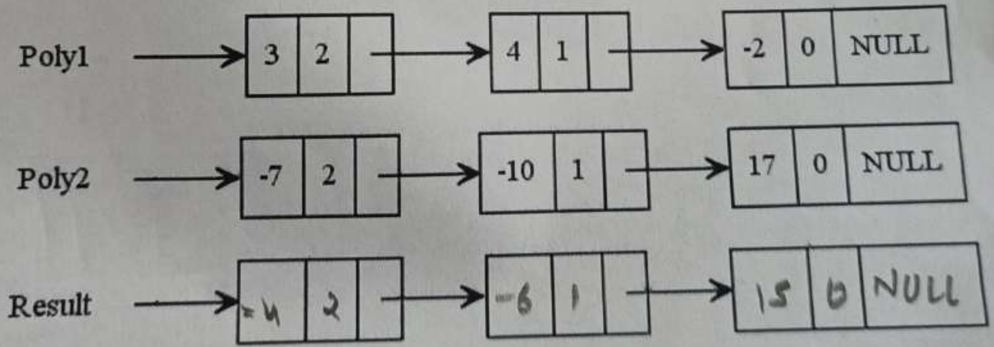
deleteMid(30)



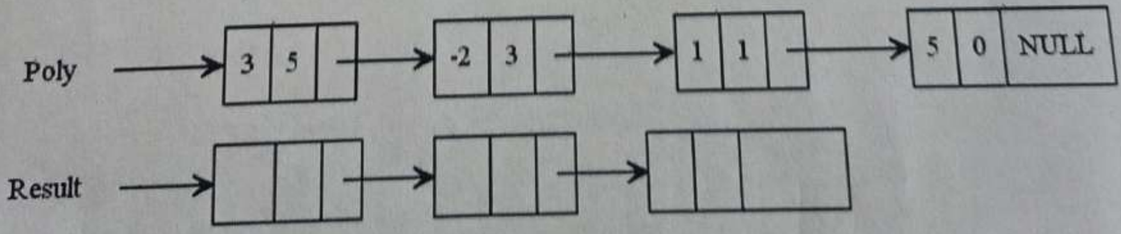
XXII. Complete me:



XXIII. Complete me:



XIV. Complete me:



```
→ struct Node* deleteAtPosition (struct Node*) {
        struct node * current = head ;
        struct node *  prev   = NULL ;
   do {
     for (int i=1 ; i<3) {
            prev = current ;
       current = current → next ;
     }
     prev → next = current → next ;
     free (current) ;
   } while (current → next != NULL);
   printf (" %d is the winner", current → data) ;
 }
```

DATA STRUCTURES AND ALGORITHMS
Unit-II-Assignment

Reg. No. : RA2211026010169 **Name** : Yatharth Patidar

Year : 2 **Branch:** CSE AIML **Section:** Y1

I. Choose the best answer:

1. Which of the following is not true for stacks?
   (a)It is a linear data structure.
   (b) It allows insertion/deletion of elements only at one end
   (c) It is widely used by systems processes, such as compilation and program control
   [✓ (d)It is based on First-In-First-Out principle]

2. Which of the following is not an example of a stack?
   (a)Collection of tiles one over another
   (b) A set of bangles worn by a lady on her arm
   [✓ (c) A line up of people waiting for the bus at the bus stop]
   (d)A pileup of boxes in a warehouse one over another

3. Tower of Hanoi can be regarded as a problem of which of the following data structures?
   [✓ (a)Stack]
   (b) Queue
   (c)Graph
   (d)Tree

4. Recursive function calls are executed using which of the following data structures?
   [✓ (a)Stack]
   (b) Queue
   (c)Graph
   (d)Tree

5. If 2, 1, 5, 8 are the stack contents with element 2 being at the top of the stack, then what will be the stack contents after following operations:
   Push (11)
   Pop ( )
   Pop ( )
   Pop ( )
   Push(7)
   (a)11, 2, 1
   (b)8, 11, 7
   [✓ (c)7, 5, 8]
   (d)5, 8, 7

6. Which of the following is best suitable for storing a simple collection of employee records?
   (a)Stack
   (b)Queue
   [✓ (c)Array]
   (d)None of the above

7. If 'top' points at the top of the stack and 'stack []' is the array containing stack elements, then which of the following statements correctly reflect the Push Operation for inserting 'item' into the stack?
   (a)top = top + 1; stack [top] = item;
   (b)stack [top] = item; top = top + 1;
   (c)stack [top++] = item;
   [✓ (d)Both (a) and (c) are correct]

8. If 'top' points at the top of the stack and 'stack []' is the array containing stack elements, then which of the following statements correctly reflect the pop operation?
   [✓ (a)top = top – 1; item = stack [top];]
   (b)item = stack [top]; top = top – 1;
   (c)item = stack [--top];
   (d)Both (b) and (c) are correct

9. If a pop operation is performed on an empty stack, then which of the following situations will occur?
   (a)Overflow
   [✓ (b)Underflow]
   (c)Array out of bound
   (d)None of the above

10. Which of the following is not a stack application?
    (a)Recursion control
    (b)Expression evaluation
    [✓ (c)Message queuing]
    (d)All of the above are stack applications

11. Which of the following statements is not true for queues?
    (a)It is a linear data structure.
    [✓ (b)It allows insertion/deletion of elements only at one end.]
    (c)It has two ends front and rear.
    (d)It is based on First-In-First-Out principle.

12. Which of the following statements is not an example of a queue?
    [✓ (a)Collection of tiles one over another.]
    (b)A queue of print jobs.
    (c)A line up of people waiting for the bus at the bus stop.
    (d)All of the above are queue examples.

13. CPU scheduler can be implemented by which of the following datastructures?
(a)Stack
✓(b)Queue
(c)Graph
(d)Tree

14. Which of the following is a type of a queue?
(a)Circular queue
(b)Priority queue
(c)Double-ended queue
✓(d)All of the above

15. If 1, 2, 3, 4 are the queue contents with element 1 at the front and 4 at the rear, then what will be the queue contents after following operations:
Insert (5)
Delete ( )
Delete ( )
Delete ( )
Insert (6)
Insert (–1)
Delete ( )
✓(a)5, 6, –1
(b)4, 5, 6, –1
(c)1, 2, 6
(d)1, 2, 6, –1

16. Which of the following is best suitable for implementing a print scheduler?
(a)Stack
✓ (b)Queue
(c)Array
(d)None of the above

17. If 'front' points at the front end of the queue, 'rear' points at the rear end of the queue and 'queue []' is the array containing queue elements, then which of the following statements correctly reflects the insert operation for inserting 'item' into the queue?
(a)rear = rear + 1; queue [rear] = item;
(b)front = front + 1; queue [front] = item;
(c)queue [rear++] = item;
✓ (d)Both (a) and (c) are correct

18. If 'front' points at the front end of the queue, 'rear' points at the rear end of the queue and 'queue []' is the array containing queue elements, then which of the following statements correctly reflects the delete operation for deleting an element from the queue?
(a)item = queue [rear]; rear = rear + 1;
(b)item = queue [front]; front = front + 1;
(c)item = queue [++front];
✓ (d)Both (b) and (c) are correct

19. If a delete operation is performed on an empty queue, then which of the following situations will occur?
    (a) Overflow
    ✓(b)Underflow
    (c)Array out of bound
    (d)None of the above

20. Which of the following is not a queue application?
    ✓(a)Recursion control
    (b) CPU scheduling
    (c)Message queuing
    (d)All of the above are queue applications

ANSWERS

1.	D	**2.**	C	**3.**	A	**4.**	A	**5.**	C	**6.**	C	**7.**	D	**8.**	A	**9.**	B	**10.**	C
11.	B	**12.**	A	**13.**	B	**14.**	D	**15.**	A	**16.**	B	**17.**	D	**18.**	D	**19.**	B	**20.**	A

II. Place me in the basket:

Stack	Queue	Double Ended Queue	Notations
PUSH	Dequeue	Enqeue Front	Infix
POP	Enqueue	EnqeueFront	Prefix
		Dequeue Front	Postfix
		Dequeue Front	

Following words are to be placed in the relevant basket:

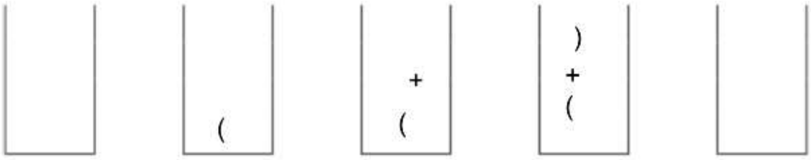
Push	Enqueue	Infix	EnqueueFront
Pop	Dequeue	DequeueRear	Prefix
Peek	DequeueFront	EnqueueRear	Postfix

III. Match me:

Column A	Column B
Infix [c]	+/ABC [a]
Prefix [a]	AB/C+ [b]
Postfix [b]	A/B+C [c]

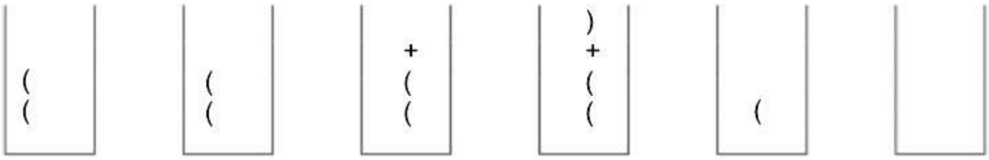
IV. Balance me:

(a+b)



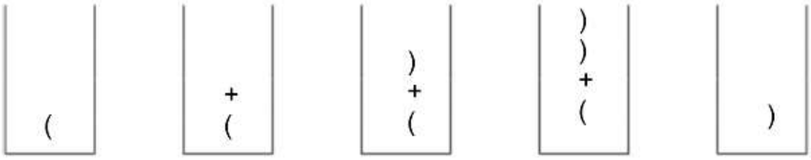
Stack is ab+ →

((a+b)



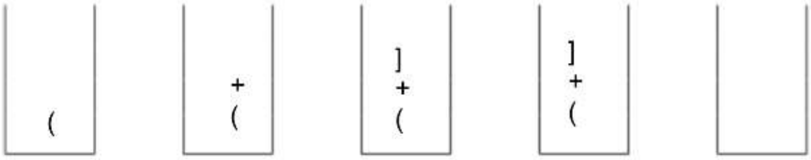
Stack is ab+ →

(a+b))



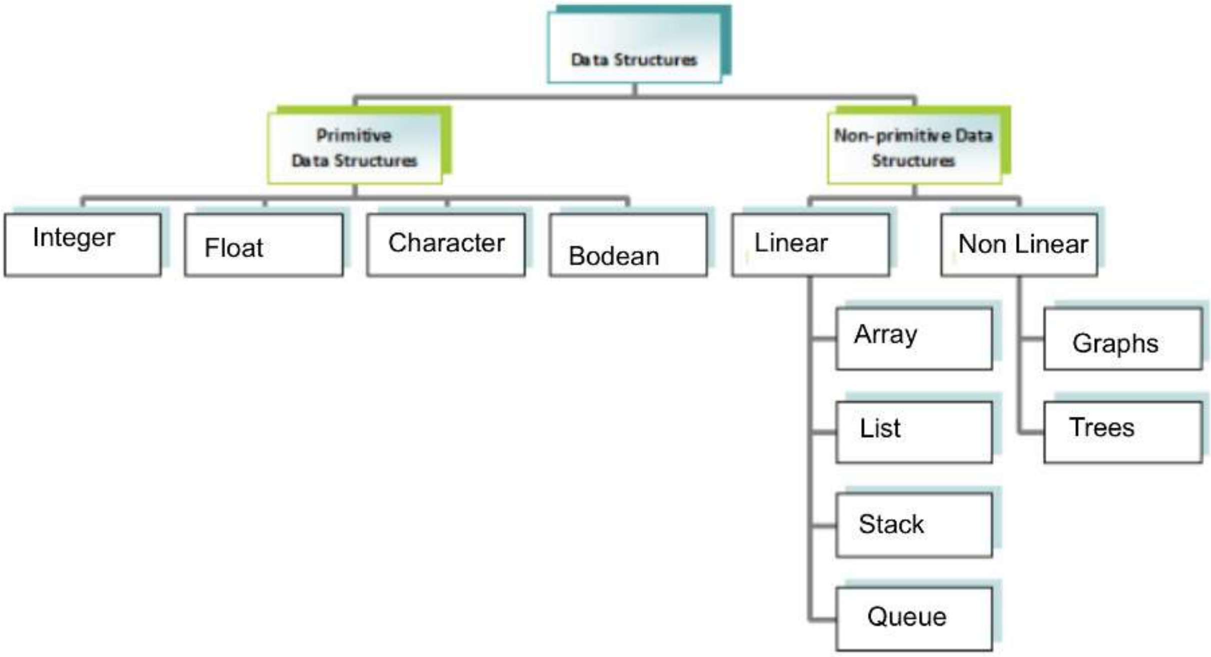
Stack is ab+ →

(a+b]



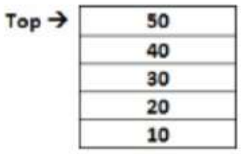
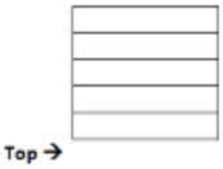
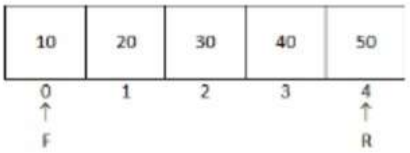
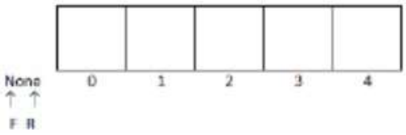
Not corresponding to the ab →

V. Complete me:

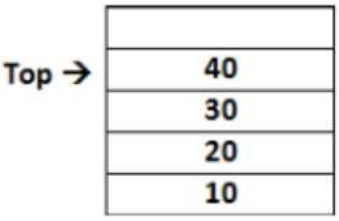
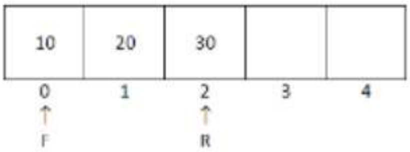


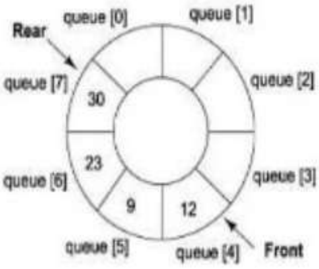
Classification of Data Structures

VI. Match me:

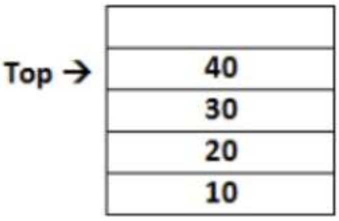
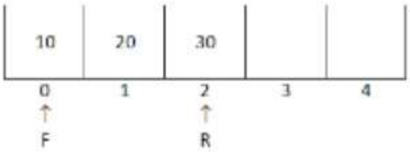
Column A	Column B
 C	Queue Overflow
 D	Queue Underflow
 A	Stack Overflow
 B	Stack Underflow

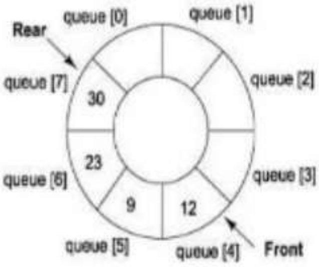
VII. Complete me:

<table>
<tr><th>Push(50)</th><th>Enqueue(50)</th><th>Enqueue(50)</th></tr>
<tr>
<td>
</td>
<td>

| 10 | 20 | 30 | 50 | |
F R</td>
<td>
</td>
</tr>
</table>

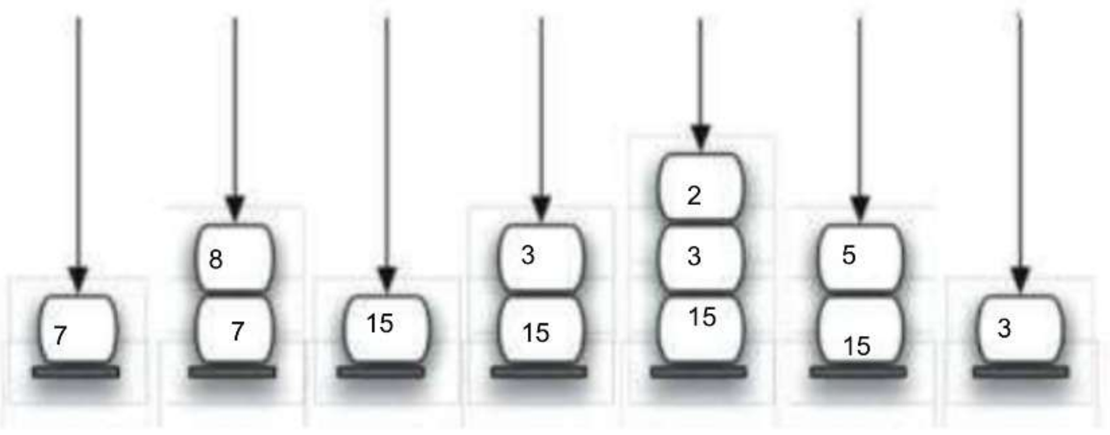
VIII. Complete me:

<table>
<tr><th>Pop()</th><th>Dequeue()</th><th>Dequeue()</th></tr>
<tr>
<td>
</td>
<td>

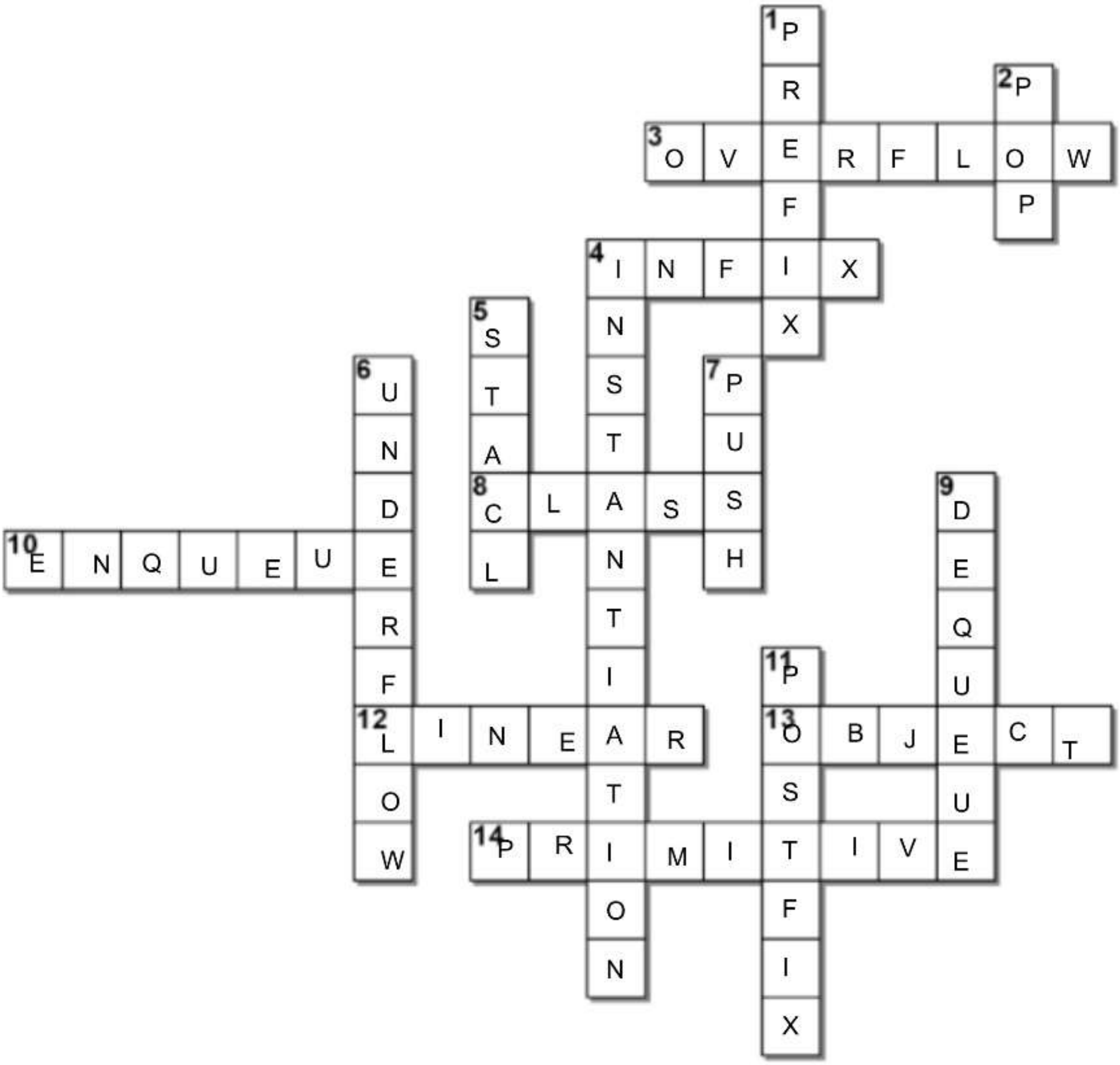
| 20 | 30 | | | |
F R</td>
<td>
</td>
</tr>
</table>

IX. Evaluate me:

7 8 + 3 2 + /



X. Fill me:



Across:

3 Attempt to insert an element when the stack is full is said to be ______.

4 In ______ notation, the arithmetic operator appears between the two operands to which it is being applied.

8 ______ is a blueprint or template for the object.

10 The process of inserting a new element on to the rear of the queue is called ______ operation.

12 In ______ data structures, all the data elements are arranged in a sequential fashion.

13 ______ is simply a collection of data (variables) and methods (functions) that act on those data.

14 ______ data structures include all the fundamental data structures that can be directly manipulated by machine level instructions.

Down:

1 In ______ notation, the arithmetic operator is placed before the two operands to which it applies.

2 The process of deleting an element from the top of stack is called ______ operation.

4 An object is also called an instance of a class and the process of creating this object is called ______.

5 A ______ is a list with the restriction that insertions and deletions can be performed in only one position, namely, the end of the list, called the top. It follows Last-In-First-Out (LIFO) principle.

6 Attempt to delete an element when the stack is empty is said to be ______.

7 The process of inserting a new element to the top of the stack is called ______ operation.

9 The process of deleting an element from the front of queue is called ______ operation.

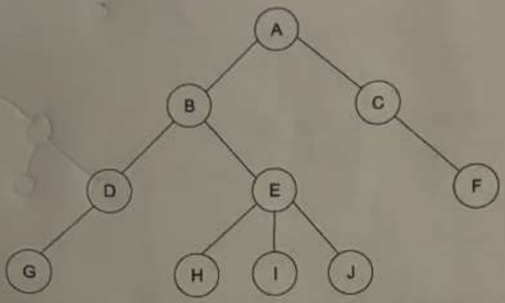
11 In ______ notation, the arithmetic operator appears directly after the two operands to which it applies.

21CSC201J - DATA STRUCTURES AND ALGORITHMS
Unit-IV-Assignment

Reg. No. : RA2211026010164 Name : Yatharth Patidar

Year : 2nd Branch: CSE - AIML Section: Y1

I. Identify me:



Root	A	Parent of D	B
Leaf Nodes	GHIJF	Depth of J	2
Siblings of B	C	Height of B	2
Degree of E	3	Depth of tree	3
Path from A to J	A-B-E-J	Height of tree	3

II. Match me:

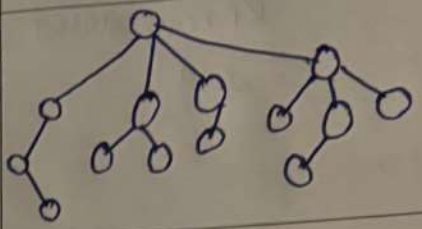
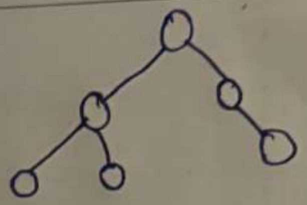
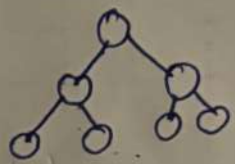
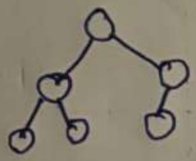
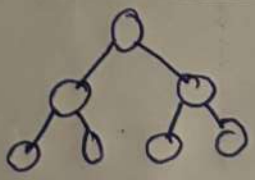
Column A	Column B
If every node in a tree has only one child. [c]	Left skew tree
If every node has only left child. [a]	Right skew tree
If every node has only right child [b]	Skew tree

III. Declare me:

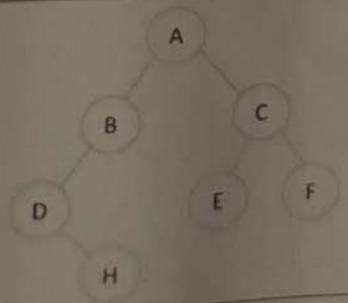
Binary Tree Node

Struct node {
int data;
struct node *left_child;
struct node *right_child;
};

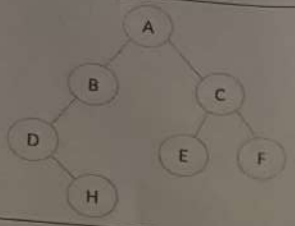
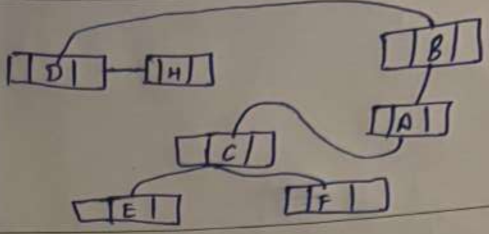
IV. Draw me:

Concept	Description	Example
General tree	General tree has any number of children.	
Binary tree	A Binary tree has not more than two children.	
Strict binary tree	A binary tree is called strict binary tree if each node has exactly two children or no children.	
Complete binary tree	A complete binary tree of height h has between 2^h and 2^{h+1} - 1 nodes. In the bottom level the elements should be filled from left to right.	
Full binary tree (or) Perfect binary tree	A full binary tree of height h has 2^{h+1} - 1 nodes.	

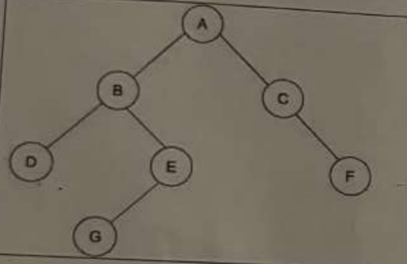
V. Represent me:

Tree	Linear
	0 1 2 3 4 5 6 - Index D | H | B | A | E | C | F

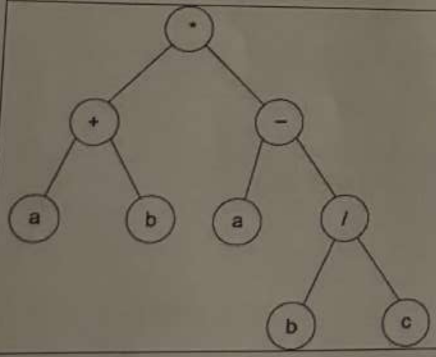
VI. Represent me:

Tree	Linked
	

VII. Represent me:

Tree	Inorder	Preorder	Postorder
	DBGEACF	ABDEGCF	DGEBFCA

VIII. Represent me:

Tree	Inorder	Preorder	Postorder
	a+b*a-b/c	*+ab-a/bc	ab+abc/-*

IX. Fill my routine:

Inorder	Preorder	Postorder

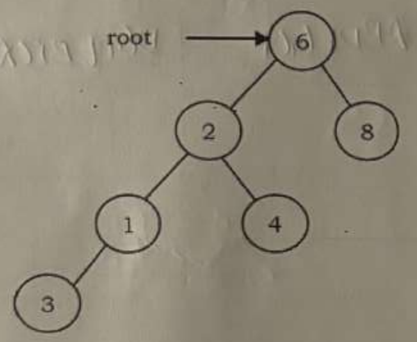
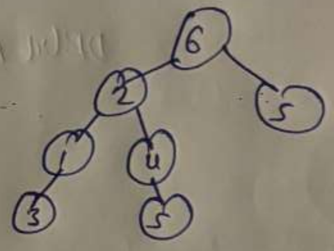
inorder(root →left);
Printf("%d", root →data);
inorder(root →right);
	Printf("%d", root →data);
	preorder(root→left);
	Preorder(root →right);
		void postorder(struct node* root)
		{ if (root == NULL)
		return;
		postorder (root→left);
		Postorder(root → right);
		Printf("%d", root→data);
		}

X. Fill my routine:

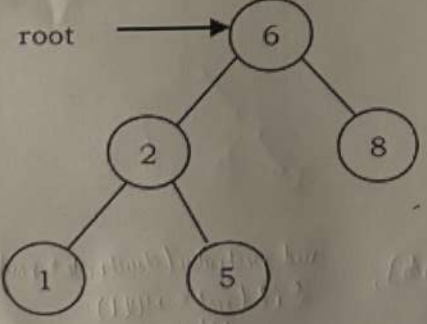
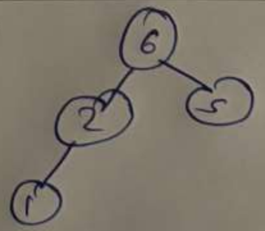
findMin (Binary Search Tree)	findMax (Binary Search Tree)

void smalle (struct node * root)
{ while (root != NULL && root->left != NULL)
{ root = root->left;
}
printf(" smalle : %d, root->data);
}
	void largest(struct node *root){
	while(root != NULL && root->right != NULL){
	root = root->right;
	}
	printf("Largest : %d", root->data);
	}

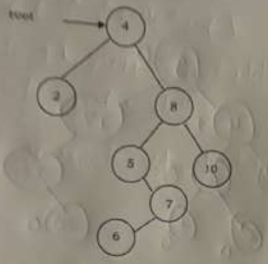
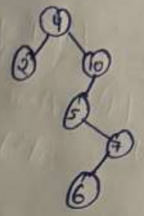
XI. Redraw me:

Binary Search Tree	insert(5)
	

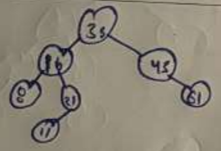
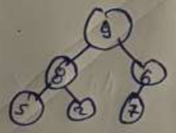
XII. Redraw me:

Binary Search Tree	delete(5)
	

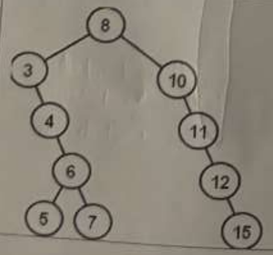
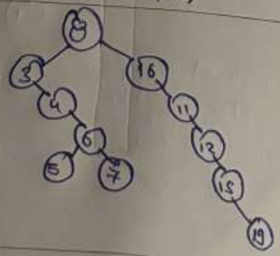
XIII. Redraw me:

Binary Search Tree	delete(8)
	

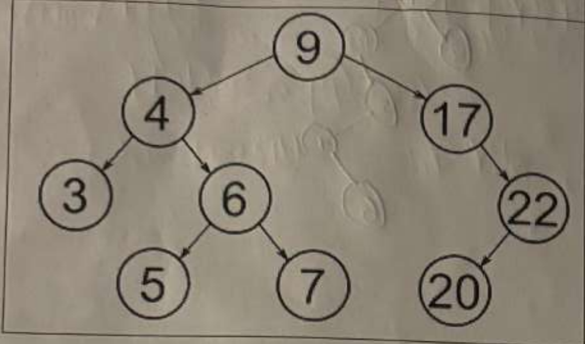
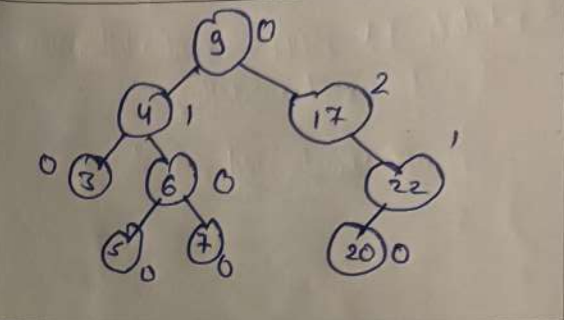
XIV. Draw me:

Concept	Description	Example
AVL Tree	An AVL Tree is self balancing binary search tree. In AVL, height of child differ by at most '1'.	
Max Haep	A max haep is complete binary tree in which value of node is >= value of its children	

XV. Redraw me:

BST Tree	insert(19)
	

XVI. Redraw me:

Binary Tree	Find Balance Factor
	

XVII. Search me:

1. Tree traversals
2. Binary search tree operations
3. Different types of binary trees
4. Representation of binary tr

F	D	P	U	J	Y	S	G	E	R	Z	O	K	J	N	X	I	J
L	R	N	U	W	J	R	Q	E	N	U	E	A	F	B	D	P	X
Q	Y	C	Z	I	B	T	R	E	S	N	I	Z	L	M	G	N	Q
O	Z	E	O	K	D	Q	Q	D	Y	H	V	L	W	P	O	X	E
F	I	N	D	M	A	X	Z	S	P	T	U	A	O	J	R	U	X
V	A	F	J	Z	P	E	L	W	Y	F	A	C	B	E	A	R	B
J	K	S	E	E	O	L	O	U	J	E	P	Z	D	R	C	S	U
E	H	I	H	P	D	U	E	W	Y	Q	M	R	K	E	L	G	E
G	M	Y	D	U	N	B	G	T	J	O	O	C	D	D	S	G	J
F	R	S	T	R	I	C	T	J	E	E	S	N	M	R	E	A	S
Z	D	S	Y	E	F	S	Y	R	R	U	T	I	O	O	N	S	D
M	E	B	C	D	O	W	Z	P	L	J	V	M	R	T	M	F	E
D	L	F	U	R	C	H	N	Z	N	W	N	D	E	S	M	X	K
N	E	W	B	O	E	Q	L	U	T	Z	T	N	U	O	Z	Y	N
H	T	A	O	N	P	X	R	M	T	B	K	I	M	P	B	B	I
U	E	I	B	I	X	R	G	H	I	D	O	F	D	X	N	S	L
B	X	P	N	I	Y	C	O	E	L	I	N	E	A	R	C	X	B
Q	R	C	Z	K	X	P	M	Q	H	M	E	P	V	H	J	K	A

XVIII. Fill me:

Across:

1 The process of visiting all nodes of a tree is called tree ______.

3 Nodes with no children are known as ______.

4 The inorder traversal of the binary tree for an arithmetic expression gives the expression in an ______ form.

5 The postorder traversal of the binary tree for the given expression gives in ______ form.

7 A tree is a collection of ______.

9 The number of subtrees of a node is called its ______.

11 In binary search tree, to perform a findMax, start at the root and go right as long as there is a ______ child.

12 In binary search tree, to perform findMin, start at the root and go left as long as there is a ______ child.

13 A tree is said to be a binary tree if it has atmost ______ children.

Down:

2 Nodes with the same parent are called ______.

5 In a tree, every node except the root has one ______.

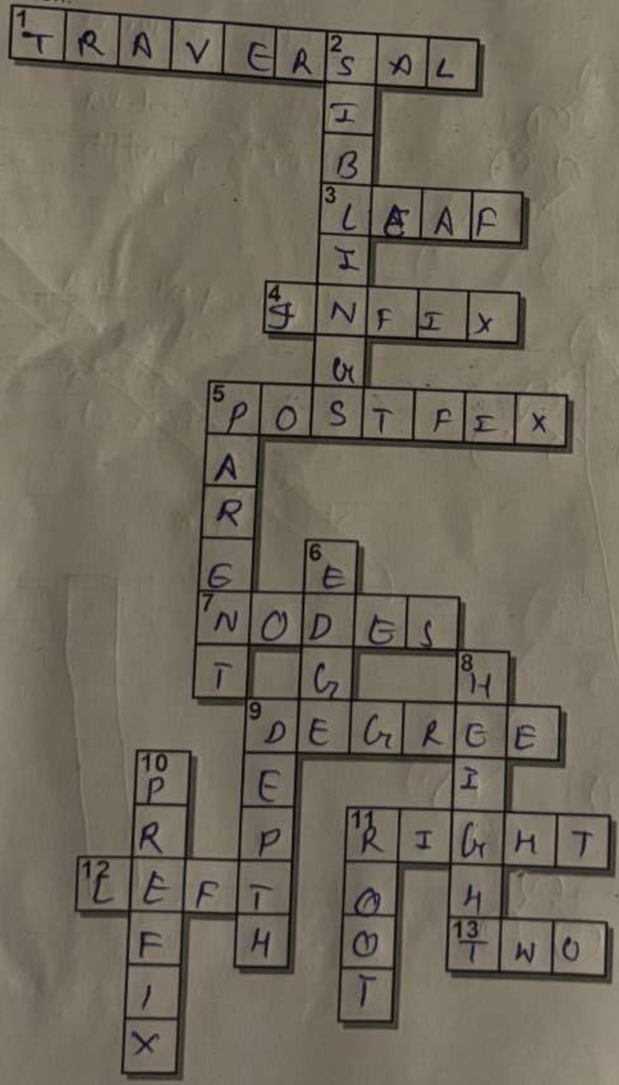
6 An ______ refers to the link from parent to child.

8 The ______ of ni is the length of the longest path from ni to a leaf.

9 For any node ni, the ______ of ni is the length of the unique path from the root to ni.

10 The preorder traversal of the binary tree for the given expression gives in ______ form.

11 The ______ of a tree is the node with no parents.



Yatharth Patidar
RA2211026010169

Unit → V

1. How many edges a complete Graph G which is undirected with 13 vertices have?
a. 156
~~b.~~ 78
c. 169
d. 84

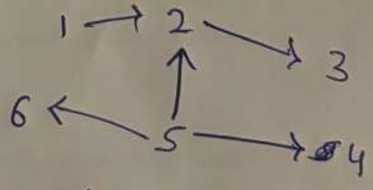
2. An edge (v,v) from a vertex to itself is called as
a. Path
b. Cycle
~~c.~~ Loop
d. Cyclic

3. Data Structure required for Breadth First Traversal is,
a. Tree
~~b.~~ Queue
c. Array
d. Stack

4. Graph is represented by
a. Adjacency Matrix
b. Adjacency List
~~c.~~ Both (a) and (b)
d. Stack or Queues

5. Given the Adjacency Matrix of a Directed Graph G, then G is a

V	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	0	1	0
4	0	1	0	0	0	0



Graph G

5	0	0	0	1	0	1
6	0	0	0	0	0	0

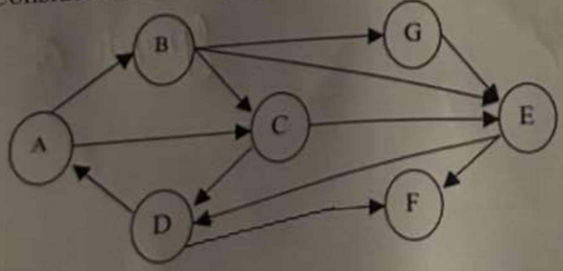
a. Directed Acyclic Graph
b. Complete Graph
[✓ c.] Cyclic Graph
d. Strongly Connected Graph

6. Given the Adjacency Matrix of a Directed Graph G, then G is a

V	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	0	0	1	1	1	0	0
3	0	0	0	0	1	0	0
4	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0

[✓ a.] Directed Acyclic Graph
b. Complete Graph
c. Cyclic Graph
d. Strongly Connected Graph

7. Consider the below graph, identify the vertex with maximum indegree.



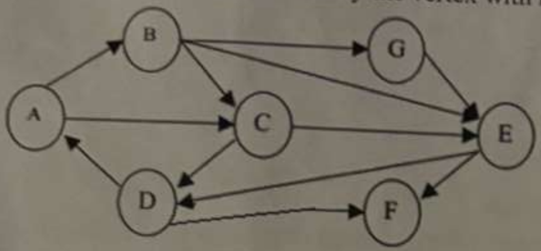
a. B

b. D

c. C

~~d.~~ E

8. Consider the below graph, identify the vertex with maximum outdegree.



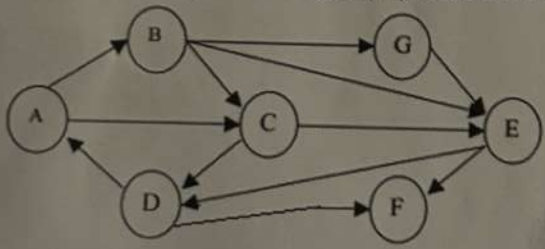
a. A

b. C

~~c.~~ B

d. E

9. Consider the below graph, identify the vertex with maximum difference. Where difference is given by, | indegree - outdegree | . (Modulus of indegree - outdegree)



~~a.~~ B

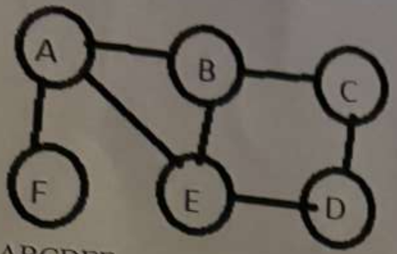
~~b.~~ F

c. A

d. D

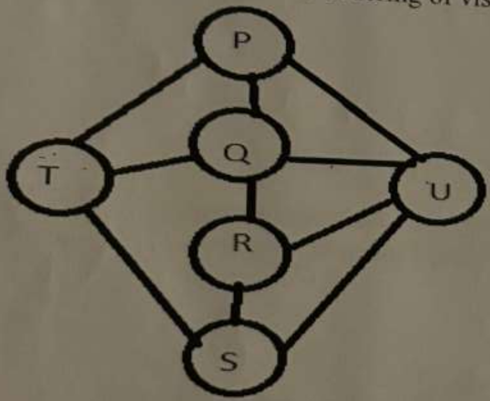
e. E

10. Consider the Graph G, Assume Breadth First Traversal is implemented in the given Graph. Identify one of the possible ordering of visiting the nodes.



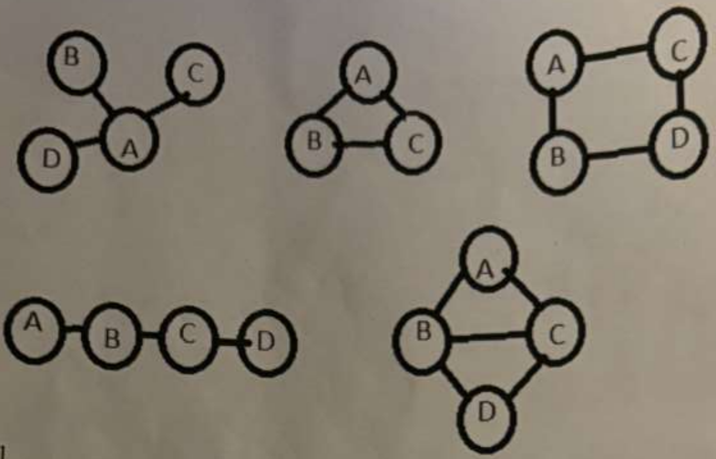
a. ABCDEF
~~b.~~ BEADCF
~~c.~~ EABDFC
d. EABDCF

11. Consider the Graph G, Assume Depth First Traversal is implemented in the given Graph. Identify one of the possible ordering of visiting the nodes.



a. PQUTRS
~~b.~~ PQURST
c. RUTSQP
d. TPRUSQ

12. Consider the below set of Graphs. Choose "A" as the starting vertex, identify how many graphs do not have the same ordering of vertex while implementing Breadth First Traversal and Depth First Traversal. (At Least one ordering should be the same in both Breadth First Traversal and Depth First Traversal).



a. 1

[✓ b. 2]

c. 3

d. 4

e. 5

13. Which is not an application of Graphs?

a. Finding Shortest Path

[✓ b. Sorting]

c. Job Scheduling

d. Web Crawlers

14. Consider a Graph G, identify true statements from the below

a. More than one vertex can have indegree 0.

b. Vertices can have same indegree and outdegree

c. There can be a vertex that has outdegree 0.

[✓ d. All statements are true.]

15. ____________identify shortest path from one node to another

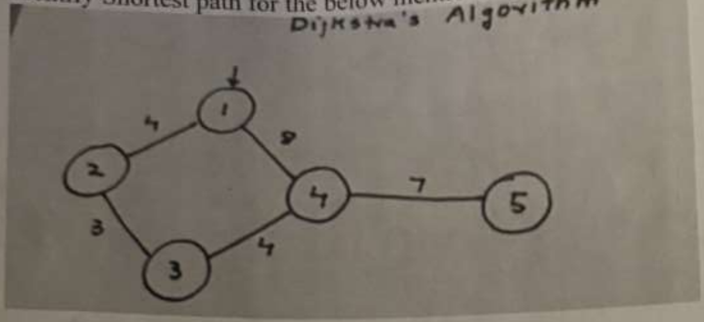
a. BFS

b. DFS

[✓ c. Dijkstra's Algorithm]

d. Topological sorting

16. Identify Shortest path for the below mentioned graph


Dijkstra's Algorithm

17. What is the routine format for dijkstra's algorithm?

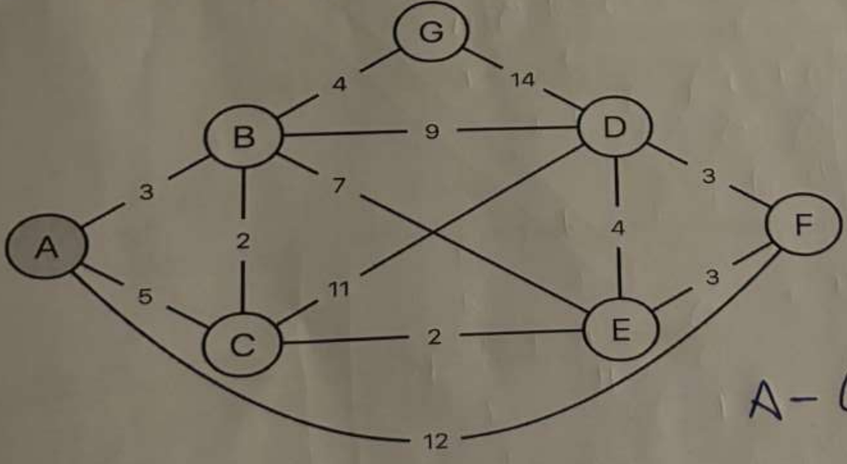
[✓ a.] if(visited[v]==0)
distance[v]=min(distance[v], distance[w]+cost[w][v])

b. if(visited[v]==1)
distance[v]=min(distance[v], distance[w]+cost[w][v])

c. if(visited[v]==0)
distance[v]=min(distance[v], distance[w]-cost[w][v])

d. if(visited[v]==2)
distance[v]=min(distance[v], distance[w]+cost[w][v])

18. What is the shortest path from A to F.



A-C-E-F

19. In Dijkstra's Algorithm the linear time supports which Data Structures

a. stack

b. heap

[✓ c.] queue

d. binary tree

20. Properties of Topological Sort are

i. DAG[Directed Acyclic Graph]

ii. Linear Ordering

iii. U->V[Vertex]

iv. Not A Graph

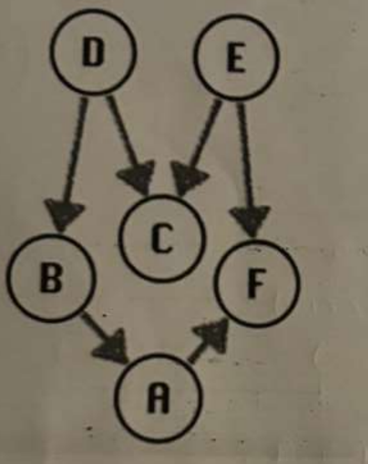
[✓ a.] i,ii

b. i,ii,iii

c. i,iii,iv

d. i,iv,ii

21.Give the Topological ordering for the following Graph



22. __________ is useful for performing a Topological sorting

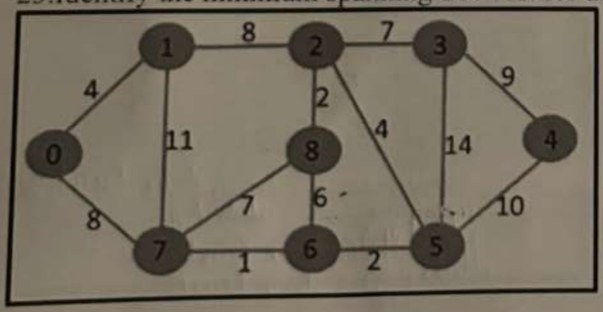
a. BFS

[✓ b.] DFS

c. Graph

d. Undirected Graph

23.Identify the minimum spanning tree cost for the given graph



24. Mention which one is not the advantage of topological sort

a. Serialization task

[✓ b.] Gives multiple Hamilton path

c. compilation task

camlin
SMOOTHER & BRIGHTER PAGES

d. Instructing scheduling

25. Identify the general properties of spanning tree

i. connected have only one spanning tree

ii. connected graph has any number of spanning trees

iii. spanning tree are maximally acyclic

iv. its Minimally connected

a. i,ii,iii

[✓ b.] ii,iii,iv

c. ii,i,iv

d. iv,i,iii