**Unit-2**

| Q.No | Question |
|---|---|
| 1. | a) Differentiate between arrays and linked lists. |

| Array | Linked list |
|---|---|
| **b) An array is a collection of elements of a similar data type.** | **c) A linked list is a collection of objects known as a node where node consists of two parts, i.e., data and address.** |
| **d) Array elements store in a contiguous memory location.** | **e) Linked list elements can be stored anywhere in the memory or randomly stored.** |
| **f) Array works with a static memory.**<br>**g) Here static memory means that the memory size is fixed and cannot be changed at the run time.** | **h) The Linked list works with dynamic memory.**<br>**i) Here, dynamic memory means that the memory size can be changed at the run time according to our requirements.** |
| **j) Array elements are independent of each other.** | **k) Linked list elements are dependent on each other.** |
| **l) Array takes more time while performing any operation like insertion, deletion, etc.** | **m) Linked list takes less time while performing any operation like insertion, deletion, etc.** |

n) Explain an algorithm to insert new node at the beginning, at middle position and at the end of a Singly Linked List.

```
Algorithm for insert at begin:
If(head==NULL)
{
head=new;
}
else
{

New->next=head;
Head=new;
}
Algorithm for insert at end:
If(head==NULL)
{
head=new;
}
else
{
Temp=head;
while(temp->next!=NULL)
{
Temp=temp->next;
}
```

```
Temp->next=new;
}

Algorithm for insert at given position:
If(head==NULL)
{
head=new;
}
else
{
printf("enter a position to insert");
scanf("%d",&pos);
temp=head;
for(i=1;i<pos-1;i++)
{
temp=temp->next;
}
temp1=temp->next;
temp->next=new;
new->next=temp1;
}
```

2. Write a c program to implement Circular Queue with the help Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
}*head,*temp,*new,*temp1;

void insert ();
void delete();
void display();
void main ()
{
   int opt;
   do
   {

      printf("\n1.Insert \n2.Delete\n3.display\n");
      printf("\nEnter your option\n");
      scanf("\n%d",&opt);
      switch(opt)
      {
         case 1:
         insert();
         break;

         case 2:
         delete();
         break;
         case 3:
         display();
         break;
```

```
        default:
        printf("Please enter valid choice.."); exit(0);
      }
   } while(1);
}
void insert()
{
   int ele;
   new = (struct node *)malloc(sizeof(struct node));

      printf("\nEnter element");
      scanf("%d",&ele);
      new->data = ele;
      if(head == NULL)
      {
         head = new;
         new -> next = head;
      }
      else
      {
         temp = head;
         while(temp -> next != head)
         {
            temp = temp -> next;
         }
         temp -> next = new;
         new -> next = head;
      }


}

void delete()
{

   if(head == NULL)
   {
      printf("\nno elements");
   }
   else if(head->next == head)
   {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }

   else
   {   temp = head;
      while(temp -> next != head)
         {
            temp = temp -> next;
         }
      temp->next = head->next;
      free(head);
      head = temp->next;
```

```
      printf("\nnode deleted\n");

   }
}


void display()
{

   temp=head;
   if(head == NULL)
   {
      printf("\nno elements");
   }
   else
   {
      while(temp -> next != head)
      {

         printf("%d-->", temp -> data);
         temp =temp -> next;
      }
      printf("%d--->", temp -> data);
      printf("%d--->", temp ->next-> data);
   }

}
```

3. a) Build a C program to traverse a given single linked list .

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
   struct node  *next;
}*new,*head,*temp,*temp1,*temp2;
void main()
{
void insert(int ele);
void delete();
void display();
int ele,opt,i;
head=NULL:
do
{
printf("1.insert     2.delete     3.display     \n");
printf("enter ur option");
scanf("%d",&opt);
switch(opt)
{
case 1:
```

```
printf("enter a value to insert");
scanf("%d",&ele);
insert(ele);
break;
case 2:
   delete();break;
case 3:
     dispay(); break;
default:
printf("wrong ption");
exit(0);
}
}while(1);//infinite loop
}
void insert(int ele)
{
int opt,pos;
new=(struct node*)malloc(1*sizeof(struct node));
new->data=ele;
new->next=NULL;
printf("1.insert begin   2.insertend   3.insert position\n");
scanf("%d",&opt);
switch(opt)
{
case 1:
    if(head==NULL)
    {
       head=new;
     }
 else
 {
   new->next=head;
   head=new;
 }
break;
case 2:
if(head==NULL)
    {
       head=new;
     }
else
{
 temp=head;
while(temp->next!=NULL)
{
```

```
temp=temp->next;
}
temp->next=new;
}
break;
case 3:
if(head==NULL)
    {
       head=new;
     }
else
{
printf("enter the position where we have to insert\n");
scanf("%d",&pos);
temp=head;
for(i=1;i<pos-1;i++)
{
temp=temp->next;
}
temp1=temp->next;
temp->next=new;
new->next=temp1;
}
break;
default: printf("wrong option");
}

void delete()
{
int  opt,pos,i;
printf("1.delete begin    2.delete end     3.delete pos\n");
printf("enter your option");
scanf("%d",&opt);
switch(opt)
{
case 1:
if(head==NULL)
{
  printf("no node to delete");
}
else
{
printf("deleted element is %d",head->data);
head=head->next;
}
```

```
break;
case 2:
if(head==NULL)
{
  printf("no node to delete");
}
else
{
temp=head;
while(temp->next->next!=NULL)
{
  temp=temp->next;
}
temp1=temp->next;
temp->next=NULL;
printf("deleted elemnent is%d",temp1->data);
}
break;
case 3:
if(head==NULL)
{
  printf("no node to delete");
}
else
{
  temp=head;
printf("enter the postion of the deleting element");
scanf("%d",&pos);
for(i=1;i<pos-1;i++)
{
temp=temp->next;
}
temp1=temp->next;
temp2=temp1->next;
temp->next=temp2;
temp1->next=NULL;
printf("deleted element is %d",temp1->data);
break;
default:  printf("wrong option");
}
}
void display()
{
temp=head;
while(temp!=NULL)
```

```
{
printf("%d--->",temp->data);
temp=temp->next;
}
```

b) Build a C function to implement insert operation in a circular linked list.

```
void insert(int ele)
{
int opt,pos;
new=(struct node*)malloc(1*sizeof(struct node));
new->data=ele;
new->next=NULL;
printf("1.insert begin   2.insertend   3.insert position\n");
scanf("%d",&opt);
switch(opt)
{
case 1:
    if(head==NULL)
    {
       head=new;
       head->next=head;
     }
  else
 {
temp=head;
while(temp->next!=head)
{
temp=temp->next;
}
temp=temp->next;
temp->next=new;
  new->next=head;
    head=new;
  }
break;
case 2:
if(head==NULL)
    {
       head=new;
head->next=head;
     }
else
{
  temp=head;
while(temp->next!=head)
```

```
{
temp=temp->next;
}
temp->next=new;
new->next=head;
}
break;
case 3:
if(head==NULL)
    {
       head=new;
     }
else
{
printf("enter the position where we have to insert\n");
scanf("%d",&pos);
temp=head;
for(i=1;i<pos-1;i++)
{
temp=temp->next;
}
temp1=temp->next;
temp->next=new;
new->next=temp1;
}
break;
default: printf("wrong option");
}
```

4. Explain an algorithm to insert new node at the beginning, at middle position and at the end of a Singly Linked List.

Algorithm for insert at begin:

```
If(head==NULL)
{
head=new;
}
else
{

new->next=head;
head=new;
}
```

Algorithm for insert at end:

```
if(head==NULL)
{
head=new;
}
else
{
```

```
temp=head;
while(temp->next!=NULL)
{
temp=temp->next;
}
temp->next=new;
}

Algorithm for insert at given position:
if(head==NULL)
{
head=new;
}
else
{
printf("enter a position to insert");
scanf("%d",&pos);
temp=head;
for(i=1;i<pos-1;i++)
{
temp=temp->next;
}
temp1=temp->next;
temp->next=new;
new->next=temp1;
}
```

5.

a)Explain the representation of polynomials using linked list with example.

**Representation of a polynomial using Linked List:**

2 3 ——→

1 0

$4x^5+2x^3+1*x^0$--

4 5 ——→

to represent polynomial expression we will require 2 data fields, one for coefficient, another for exponent

node format:

coeff expo next

ex: $4x^5+2x^3+1*x^0$--

structure representation polynomial:

```
struct node
{
   int coef
   int exp;
   struct node *next;
}*new,*temp,*head;
```

b) Build an algorithm for polynomial addition using linked list

p1---$4x^5+2x^3+3x$

p2-$2x^4+9x^3+5x^2$

p3=p1+p2=$4x^5+2x^4+11x^3+5x^2+3x$

```
p1->exp>p2->exp

p1->exp<p2->exp

p1->exp==p2->exp

if(p1->exp>p2->exp)
{
p3->coef=p1->coef;
p3->exp=p1->exp;
p=p1->next;
}
if(p1->exp<p2->exp)
{
p3->coef=p2->coef;
p3->exp=p2->exp;
p2=p2->next;
}
if(p1->exp==p2->exp)
{
p3->coef=p1->coef+p2->coef;
p3->exp=p1->exp;
p1=p1->next;
p2->p2->next;
}
```

6. a)Explain the representations of sparse matrix with example.

**Sparse Matrix representation using linked list:**

sparse matrix is a one which having very few Non-zero elements

i.e if we consider a matrix of size M*N--then the memory required to store values is

M*N*S(size of int--no of bytes required to store a single value)

int a[5][6];---memory required is--5*6*2=60bytes

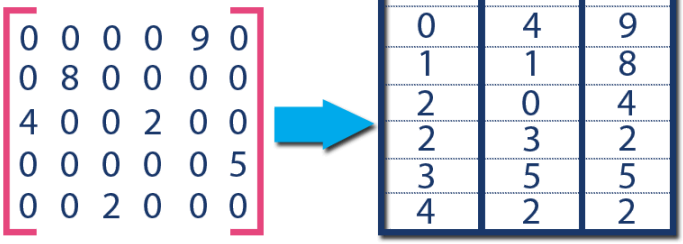
here in our example 6 non zero elements---6*2=12bytes

while we are performing searching requires more Time and space complexity

Representation of Sparse matrix using arrays:(6*3*2--18*2=36bytes)

for representing sparse matrix w will use triplet(3 columns) which consists

1.row no  2. coloum no  3.value(element)



| Rows | Columns | Values |
|---|---|---|
| 5 | 6 | 6 |
| 0 | 4 | 9 |
| 1 | 1 | 8 |
| 2 | 0 | 4 |
| 2 | 3 | 2 |
| 3 | 5 | 5 |
| 4 | 2 | 2 |

/write a c-program to represent Sparse matrix using Linked List:

```
#include<stdio.h>
#include<stdlib.h>
strcut node
{
int row;
int col;
int data;
struct node *next;
}*head,*new,*temp;
void main()
{
void insert(int,int,int);
void display();
int i,j,a[5][6],ele;
head=NULL;

printf("enter the elements in to sparse matrix\n");
for(i=0;i<5;i++)
{
for(j=0;j<6;j++)
{
  scanf("%d",&a[i][j]);
}
}
```

```
for(i=0;i<5;i++)
{
for(j=0;j<6;j++)
{
if(a[i][j]!=0)
{
insert(i,j,a[i][j]);
}
}
}
display();
}
void insert(int i,int j,int val)
{
new=(struct node *)malloc(1*sizeof(struct node);
new->row=i;
new->col=j;
new->data=val;
new->next=NULL;
if(head==NULL)
{
head=new;
}
else
{
temp=head;
while(temp->next!=NULL)
{
  temp=temp->next;
}

temp->next=new;
}
}
void display()
{
temp=head;
while(temp->next!=NULL)
{
printf("%d    %d    %d\n",temp->row,temp->col,temp->data);
temp=temp->next;
}
printf("%d    %d    %d\n",temp->row,temp->col,temp->data);
}
```

b)What are the advantages and disadvantages of singly linked list?

Advantages:

| Linked list |
| --- |
| A linked list is a collection of objects known as a node where node consists of two parts, i.e., data and address. |
| Linked list elements can be stored anywhere in the memory or randomly stored. |
| The Linked list works with dynamic memory.<br>Here, dynamic memory means that the memory size can be changed at the run time according to our requirements. |
| Linked list takes less time while performing any operation like insertion, deletion, etc. |

Dis advantages:

1.Memory usage: More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.

2.Traversal: In a Linked list traversal is more time-consuming as compared to an array. Direct access to an element is not possible in a linked list as in an array by index.

7.

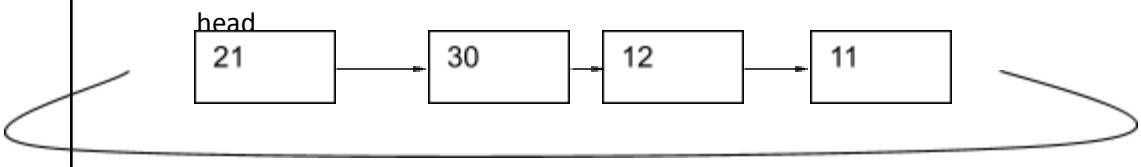
a) Construct the stacks using linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
   struct node  *next;
}*new,*head,*temp,*temp1,*temp2;
void main()
{
void push(int ele);
void pop();
void display();


int ele,opt,i;
head=NULL;
do
{
printf("1.push     2.pop    3.display      \n");
printf("enter ur option");
scanf("%d",&opt);
```

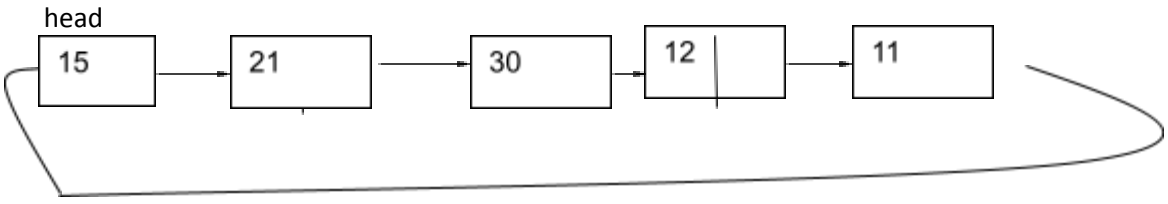
```
switch(opt)
{
case 1:
printf("enter a value to insert");
scanf("%d",&ele);
push(ele);display();
break;
case 2:
   pop();display();break;
case 3:
     display(); break;
default:
printf("wrong ption");
exit(0);
}
}while(1);//infinite loop
}
void push(int ele)
{
int opt,pos,i;
new=(struct node*)malloc(1*sizeof(struct node));
new->data=ele;
new->next=NULL;
if(head==NULL)
    {
       head=new;
     }
else
{
  temp=head;
while(temp->next!=NULL)
{
temp=temp->next;
}
temp->next=new;
}
}
void pop()
{
  int opt,pos,i;

if(head==NULL)
{
printf("deletion is not possible:");
}
```

```
temp=head;
while(temp->next->next!=NULL)
{
  temp=temp->next;
}
temp1=temp->next;
temp->next=NULL;
printf("deleted element is %d:",temp1->data);
}
void display()
{
 temp=head;
while(temp!=NULL)
{
printf("%d---->",temp->data);
temp=temp->next;
}
}
```

With a neat diagram represent 4 elements ( 21, 30, 12, 11) in Circular linked list and explain how to insert 15 at beginning ?




To insert 15 at begining:
1.create a new node with data value as 15
2.temp=head
3.move temp position to last node
4.then tem->next=new;
5.head=new;



8

a) Describe how a node can be deleted at a user specified position in a doubly linked list

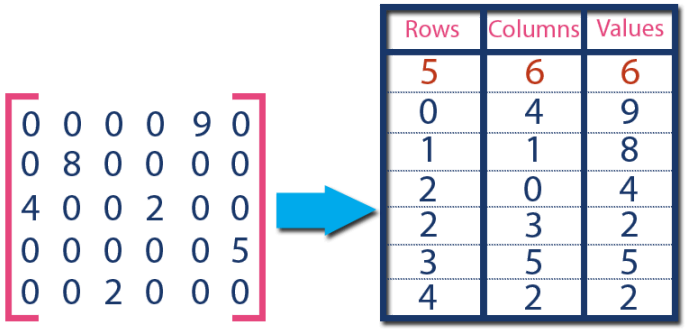
```
void delete(struct node *head)
{
if(head==NULL)
{
  printf("no node to delete");
}
else
{
  temp=head;
printf("enter the postion of the deleting element");
scanf("%d",&pos);
for(i=1;i<pos-1;i++)
{
temp=temp->next;
}
temp1=temp->next;
temp2=temp1->next;
temp->next=temp2;
temp2->prev=temp;
temp1->next=NULL;
temp1->prev=NULL;
printf("deleted element is %d",temp1->data);
}
```

b)Write C structure to declare node of a double-linked list. Discuss the advantages and limitations of linked lists.

```
struct node
{
     int data;
     struct node  *prev;
     struct node  *next;
}*new,*head,*temp,*temp1,*temp2;
```

advantages:

| Linked list |
| --- |
| A linked list is a collection of objects known as a node where node consists of two parts, i.e., data and address. |
| Linked list elements can be stored anywhere in the memory or randomly stored. |
| The Linked list works with dynamic memory. |

| | |
|---|---|
| | Here, dynamic memory means that the memory size can be changed at the run time according to our requirements.<br>Linked list takes less time while performing any operation like insertion, deletion, etc.<br>advantages:<br>1.Memory usage: More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.<br>2.Traversal: In a Linked list traversal is more time-consuming as compared to an array. Direct access to an element is not possible in a linked list as in an array by index. |
| 9 | a) Explain with an example, how linked listsand arrays can be used for sparse matrix representation.<br>**Sparse Matrix representation using linked list:**<br>sparse matrix is a one which having very few Non-zero elements<br>i.e if we consider a matrix of size M*N--then the memory required to store values is<br>M*N*S(size of int--no of bytes required to store a single value)<br>int a[5][6];---memory required is--5*6*2=60bytes<br>here in our example 6 non zero elements---6*2=12bytes<br>while we are performing searching requires more Time and space complexity<br>Representation of Sparse matrix using arrays:(6*3*2--18*2=36bytes)<br>for representing sparse matrix w will use triplet(3 columns) which consists<br>1.row no 2. coloum no 3.value(element)<br>  <br>/write a c-program to represent Sparse matrix using Linked List:<br>#include<stdio.h><br>#include<stdlib.h><br>strcut node<br>{<br>int row;<br>int col;<br>int data; |

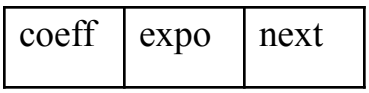
```
struct node *next;
}*head,*new,*temp;
void main()
{
void insert(int,int,int);
void display();
int i,j,a[5][6],ele;
head=NULL;

printf("enter the elements in to sparse matrix\n");
for(i=0;i<5;i++)
{
for(j=0;j<6;j++)
{
  scanf("%d",&a[i][j]);
}
}
for(i=0;i<5;i++)
{
for(j=0;j<6;j++)
{
if(a[i][j]!=0)
{
insert(i,j,a[i][j]);
}
}
}
display();
}
void insert(int i,int j,int val)
{
new=(struct node *)malloc(1*sizeof(struct node);
new->row=i;
new->col=j;
new->data=val;
new->next=NULL;
if(head==NULL)
{
head=new;
}
else
{
temp=head;
while(temp->next!=NULL)
{
```

```
  temp=temp->next;
}

temp->next=new;
}
}
void display()
{
temp=head;
while(temp->next!=NULL)
{
printf("%d    %d    %d\n",temp->row,temp->col,temp->data);
temp=temp->next;
}
printf("%d    %d    %d\n",temp->row,temp->col,temp->data);
}
```

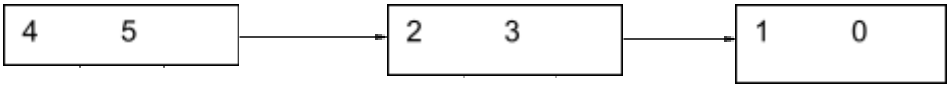
b) Explain how a polynomial is represented using linked list.

To represent polynomial expression we will require 2 data fields, one for coefficient, another for exponent and one address field

node format:




ex: $4x^5+2x^3+1*x^0$



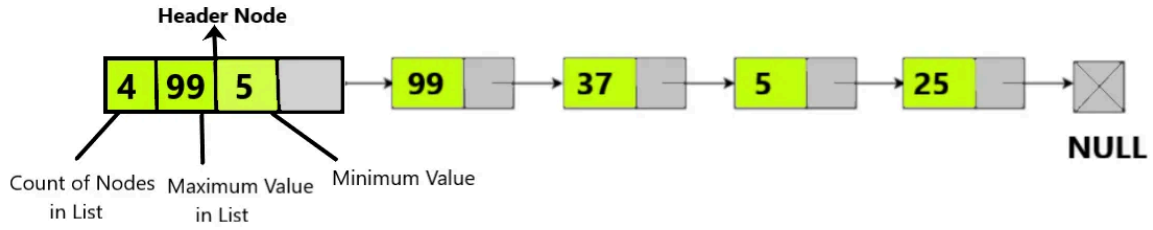

structure representation polynomial:

```
struct node
{
   int coef
   int exp;
   struct node *next;
}*new,*temp,*head;
```

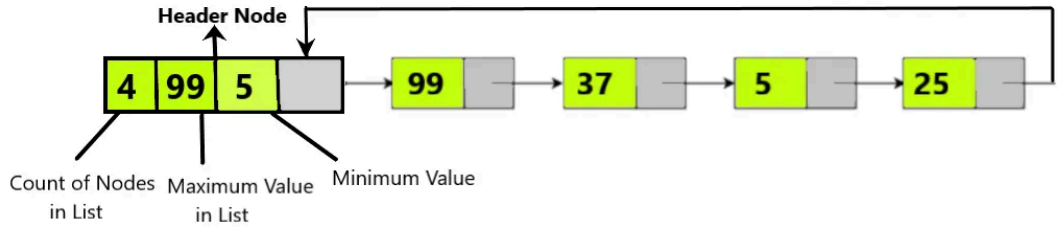
| | |
|---|---|
| 10 | a) Define a header linked list. |

- Header Linked List is a modified version of Singly Linked List. In Header linked list, we have a special node, the Header Node present at the beginning of the linked list.
- The Header Node is an extra node at the front of the list storing meaningful information about the list.

Header Linked List is of two types:
1.Grounded Header Linked List
2.Circular Header Linked List.
1.Grounded Header Linked List
In this type of Header Linked List, the last node of the list points to **NULL** or holds the reference to NULL Pointer




2.Circular Header Linked List
A Linked List whose last node points back to the First node or the Head Node of the list is called a Circular Linked List




b) Implement a C program to implement queues using linked lists.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int data;
   struct node  *next;
}*new,*head,*temp,*temp1,*temp2;
void main()
{
void insert(int ele);
void delete();
void display();
int ele,opt,i;
head=NULL:
do
{
```

```
printf("1.insert    2.delete    3.display    \n");
printf("enter ur option");
scanf("%d",&opt);
switch(opt)
{
case 1:
printf("enter a value to insert");
scanf("%d",&ele);
insert(ele);
break;
case 2:
   delete();break;
case 3:
     dispay(); break;
default:
printf("wrong ption");
exit(0);
}
}while(1);//infinite loop
}
void insert(int ele)
{
new=(struct node*)malloc(1*sizeof(struct node));
new->data=ele;
new->next=NULL;
if(head==NULL)
    {
       head=new;
     }
else
{
  temp=head;
while(temp->next!=NULL)
{
temp=temp->next;
}
temp->next=new;
}
}

void delete()
{
if(head==NULL)
{
  printf("no node to delete");
```

```
}
else
{
printf("deleted element is %d",head->data);
head=head->next;
}
}
void display()
{
temp=head;
while(temp!=NULL)
{
printf("%d--->",temp->data);
temp=temp->next;
}
```