```
        break;
    }i++
}while (expr[i] != '\0');
    return stk[t];
}
```

do{
——
}while();

Queues:

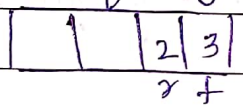
ans → Queue is a data structure where elements are inserted from one end and elements are deleted from the other end.
The end at which new elements are inserted are called rear end. & The end from which elements are deleted are called as front end.

Q. Develop a c program Queue using array :-
[Linear Queue]

```
#include <stdio.h>
# define maxsize 5
int f, r, q[10];
void main ()
{
  int choice;
  f = 0; r = -1;
for(;;)
{
  printf ("1. Insert \n 2. delete \n display");
  printf ("\n Enter the choice");
  scanf ("%d", & choice);
```

```
Void display ()
{
 if (f>r)
{
 printf ("Queue is Empty");
  exit(0);
}
printf("contents of queue are");
for (i=f ; i<=r; i++)
{
   printf("%d", q[i]);
}
}
```

Disadvantage of Queue



The sitition as shown above where it doesn't allow you to insert the Elements into the queue even if the Queue is empty.

ADT of Queue:

ADT Queue is
Object : A finite ordered list of which Zero or more Elements
functions : for all queue ∈ queue, item elements, maxQueueSize ∈ positive integer

- Queue create Q (maxQueueSize) ::= create an empty queue whose maximum size is maxQueuesize

- Boolean is full (queue, maxQueue size) ::=
  if (no. of elements in queue == max Queue size)
  return TRUE
  else return FALSE
- Queue AddQ (queue Item) ::=
  if (isfull (queue)) Queue full
  else insert item at rear of queue &
  return Queue.
- Boolean is emptyQ (queue) ::=
  if (queue == creatQ (max queue size))
  return TRUE
  else return FALSE.
- Element DeleteQ (queue) ::=
  if (is empty (queue)) return
  else remove & return @ item at front of queue.

Queue Applications :-

1. Write a C-programm to stimulate the working of Messaging system in which a message is placed in a Queue by a message sender, a message is removed from the queue by a message Receiver, which can also display the contents of the Queue.

```
#Include <stdio.h>
#Include <string.h>
# define size 5
int f, r, q[size];
Struct queue
{
```

```
    int id;
    Char msg[30];
};
q[size];
void main()
f=0; r=-1
for(;;)
{
    Printf("1: Insert\n 2: Delete\n 3: Display\n 4: Exit\n");
    printf("Enter your choice")
    Scanf("%d", &choice);
Switch(choice)
{
```

```
void insert()
{
    int x;
    char txt[20];
    if (r == size-1)
{
    printf("Queue is full\n");
      exit(0);
}
printf("Enter the id & message\n");
scanf("%d", &x);
    gets(txt);
r++;
q[r].id = x;
strcpy(q[r].msg, txt);
Printf("%d msg is inserted\n", x);
}
```

GoodLuck | Page No.
Date

```
void delet ()
{
  if (f>r)
  {
    printf ("Queue is empty\n");
    f=0 ; r=-1;
    return;
  }
  printf ("%d is deleted\n", q[f++].id);
}

void display()
{ int i;
  if (f>r)
{
  printf ("Queue is empty");
  exit(0);
}
  for (i=f; i<=r; i++)
  printf ("%d\t%s\n", q[i].id, q[i].msg);
  printf ("\n");
}
```

Chapter-2
Second part

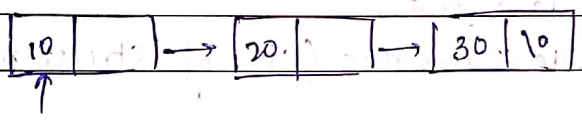
GoodLuck | Page No.
Date

Linked List.

Linked list is a collection of zero or more nodes where each node is connected to one or more nodes. Each node has 2 fields i.e

→ info field
→ link field

pictorial representation of link list.
10, 20, 30.



Advantages of linked list with arrays.

(1) The size of the array is fixed whereas linked list is dynamic.
(2) Insertion & deletion operation involving array is tdr job but its easy in case of linked list.

Types of linked list.

1] singly linked list
2] doubly linked list
3] circular singly linked list.
4] circular doubly linked list.

GoodLuck | Page No.
Date

Singly linklist :-

If there exist only one link field in each node of the list. The link list is called singly linked list

Operations

```
Struct node
{
    int data;
    Struct node * link;
};
Struct node * first;
first = (Struct node *) malloc (size of (Struct node));
```

Self referential structure (ptr → struct node)

use ?
* 16 bits - 2 bytes
32 bits - 4 bytes

Operations of Singly Linked list :

1. Inserting a node into singly linked list
2. Deleting a node from the singly linked list
3. Search a singly linked list
4. Display the contents of SLL.

INSERTING A NODE INTO SLL

(a) Inserting a node at the end of the list
(b) Inserting " " " front " " "
(c) Inserting " " " given posⁿ in the list.

GoodLuck | Page No.
Date

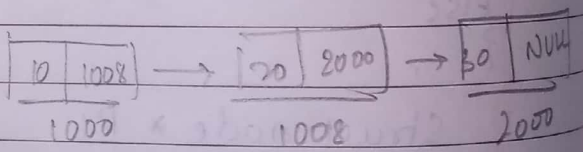
a]

```
Struct node
{
  int data;
  Struct node * link;
};
Struct node * first = NULL;
Void addend()
{
 Struct node * temp;
temp = (struct node *) malloc (sizeof (struct node));
printf ("enter element");
Scanf ("%d", & temp → data);
temp → link = NULL;
 if (first == NULL)
{
   first = temp;
}
else
{
   Struct node * P;
    P = first;
while (P → link != NULL)
{
  P = P → link;
}
  P → link = temp.
}
```

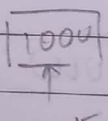
b]

```
Struct node
{
   int data;
   Struct node * link;
};
Struct node * first = NULL;
Void addbegin()
{
   struct = (struct node *) malloc (sizeof (Struct node));
   printf("Enter Elements");
   Scanf("%d", &temp → data);
   temp → link = NULL;
   temp → link = first;
      first = temp;
}
```

c] Finding length of Singly linked list.




```
Struct node
{
   int data;
   Struct node * link;
};
 Struct node * first
void length()
{
     int count=0;
    Struct node * temp;
      temp = first;
while (temp != NULL)
{
   count++;
   temp = temp → link;
}
```



GoodLuck | Page No.
Date

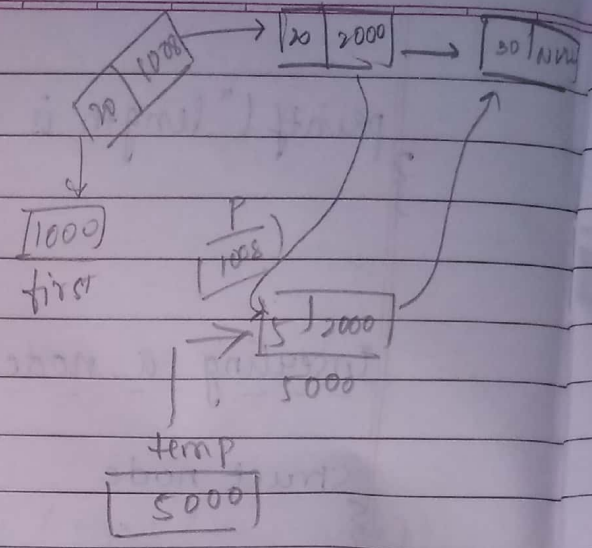
```
printf("length is %d", count);
}
```

Inserting a node at a given postn :-

```
Struct node
{
  int data;
Struct node * link;
};
Struct node * first
int length()
{
  int count=0;
 Struct node * temp;
  temp = first;
while ( temp != NULL)
{
  Count ++;
  temp = temp -> link;
}
printf("length is %d", count);
Return (count);
}
Void insertafter()
{
  Struct node *p *temp;
   int len, loc, i=1;
printf("Enter location");
Scanf("%d", & loc)
len = length();
```

GoodLuck | Page No.
Date

```
if (loc > leng)
{
  printf (" Invalid location");
}
  else
{
   p = first
  while (i<loc)
{
   p = p → link;
    i++;
}
temp = (struct node *) malloc (sizeof (structnode))
printf ("Enter the Element")
Scanf ("%d", & temp → data);
temp → link = NULL;
temp → link = p → link;
 p → link = temp;
}
```



* Display the contents of Singly linked list

```
Void display()
{
    Vat Struct node * temp;
        temp = first;
  if (temp == NULL)
{
   printf (" list is empty");
    exit(0);
}
else {
 while ( temp != NULL)
```

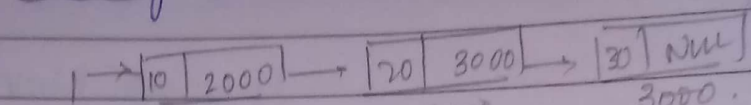
```
        printf("%d", temp->data);
        temp = temp->link;
    }
}
```

* Searching a singly linked list

```
void search()
{
    int element;
    struct node *temp;
    temp = first;
    if (temp == NULL)
    {
        printf("list is empty");
        exit(0);
    }
    else
    {
        printf("Enter element to be searched");
        scanf("%d", &element);
        while (temp != NULL)
        {
            if (temp->data == element)
            {
                printf("%d element is found", element);
                exit(0);
            }
            temp = temp->link;
```

Date

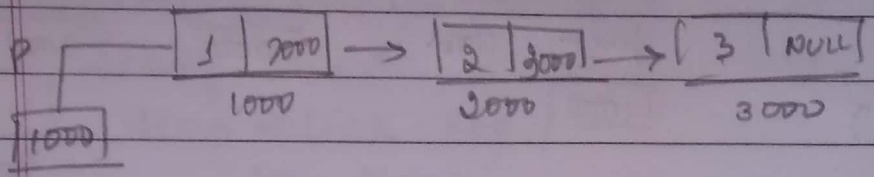
Deleting nodes in a linked list

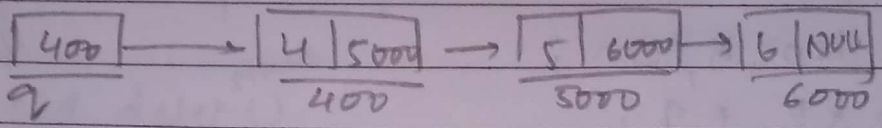



```
void delet_link()
{
  Struct node *temp, *p, *q;
  int loc, len, i=1;
  printf("Enter location");
  Scanf("%d", &loc);
   len = length()
  if (loc > len)
   printf("invalid location");
  else if (loc==1)
{
    temp = first;
    first = temp->link;
    temp->link = NULL;
    free (temp);
}
  else
{
   p = first;
  while (i<loc-1)
{
    p = p->link;
     i++;
}
  q = p->link;
  p->link = q->link;
  q->link = NULL;
  free (q);
}
```

GoodLuck | Page No.
Date

Merging of two linked list.

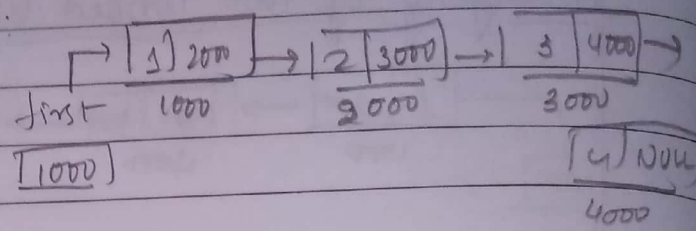





```
Struct node * merge()
{
Struct node * temp;
   temp = p;
While ( temp != NULL)
{
  temp = temp -> link
}
temp -> link = q;
   q = NULL;
  free (q);
return p;
}
```

GoodLuck | Page No.
Date

Reversing the linked :




```
void reverse()
{
 int i, j, len, k, tem;
 Struct node *p, *q;
 len = length()
  i = 1;
  j = len;
 p = q = first
  while (i<j)
{
   k = 1;
   while (k<j)
 {
    q = q→link;
     k++;
  }
```

* Implementation of stack using singly linked list.

```
Void insert()
{
 Struct node
{
   int data;
   struct node * link;
};
 Struct node * top = NULL;
void insert()
{
   Struct node * temp;
```

```
    temp = (struct node *) malloc (sizeof (struct node));
    printf ("Enter Element to be inserted");
    scanf ("%d", &temp->data);
      temp->link = top;
        top = temp;
}
void delet()
{
  if (top == NULL)
  printf ("stack is empty");
  else
  {
   psintf ("%d Element is deleted");
    top = top->link;
  }
}
void display()
{
  struct node *temp;
  if (top == NULL)
  printf ("Stack is empty")
  else { temp = top;
   while (top != NULL)
   {
    printf ("%d", top->data);
    top++; temp = top->link;
   }
  }
}
```

Implementation of Queue using Singly Linked list

```
struct node
{
  int data;
  struct node * link;
};
  struct node * front = NULL;
  struct node * rear = NULL;
void insert()
{
  struct node * temp;
  temp = (struct node *) malloc (size of (struct node));
 printf (" enter Element to be inserted ");
  scanf (" %d", & temp → data);
     temp → link = NULL;

    if ( rear = = NULL)
   {
       rear = temp;
       front = temp;
   }
  else
    {
      rear → link = temp;
       rear = temp;
    }
void deletequeue()
{
 if ( front = = NULL)
 printf (" queue is empty");
 else
 {
```

```
printf ("deleted elements is %d", front -> data);
  front = front -> link;
 }
}.

void display()
{
  struct node * temp;
 if ( front == NULL || rear == NULL)
  printf ("Queue is empty");
 else
 {
   temp = front
   while (temp != NULL)
   printf ("%d", temp -> data);
    temp = temp -> link;
  }
 }
}
```

Implementing Circular Queue using Arrays :-

```
# include <stdio.h>
# define maxsize 5
 int f, r, q[10];
void main()
{
  int choice;
  f=0; r=-1;
  for(;;)
{
  printf("1. Insert\n 2. delete\n display");
  printf("enter the choice");
  scanf("%d", &choice);
```

```
switch (choice)
{
  case I : insert ;
           break;
  case II : delet();
           break;
  case III : display();
           break;
}
}
}
  int count=0;
void insert();
{
  if (count == maxsize)
  printf ("queue is full");
else
{
  printf ("enter Element");
  Scanf ("%d", &item);
  rear = (rear+1) % maxsize;
   q[rear] = item;
   count++;
}}
void deletqueue()
{
  if (count = 20)
printf ("queue is empty");
else
{
  printf ("delete element is %d", q[front]);
  front = (front+1) % maxsize;
  count--;
}}
```

```
void display()
{
 if (count==0)
printf("Queue is empty");
else
{
 printf("Contents of Queue are");
 for(i=front ; i<=rear ; i++)
printf("%d", q[i]);
}
}
```

Implementation of circular Singly linked list.

If the link field of last nodes contains addr. of the 1st node in the list is called as circular list.

first
tail=NULL

```
void insert()
{
 Struct node *temp;
 temp = (struct node) malloc (sizeof(struct node));
 printf("Enter Element to be inserted");
 Scanf("%d", &temp->data);
  temp->link = NULL;
 if (first==NULL)
{
   first = temp;
   tail = temp;
}
else
{
   tail->link = temp
   tail= temp;
   tail->link = first;
```

```
switch (choice)
{
case 1 : insert ;
        break;
case 2 : delete ();
        break;
case 3 : display ();
        break;
}
}

void insert()
{
  int item;
  if (r == maxsize - 1)                    size - 1
{
  printf (" queue is full");
   exit(0);
}
printf (" Enter the elements ");
scanf (" %d", item);
   r = r + 1;
   i[r] = item;
}


void delet()
{
   if ( f > r)
{
  printf (" Queue is Empty");
   exit(0);
}
printf (" Deleted element is %d \n", q[f]);
        f = f + 1 ;
}
```