LAB PROGRAMS- 5,6,7

1. WAP to Implement Singly Linked List with following operations
a)Create a linked list.b)Insertion of a node at first position, at any position and at end of list.c) Display the contents of the linked list.

SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
struct node
{
   int info;
   struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
   NODE x;
   x = (NODE)malloc(sizeof(struct node));
   if (x == NULL)
   {
      printf("mem full\n");
      exit(0);
   }
   return x;
}
void freenode(NODE x)
{
   free(x);
}
NODE insert_front(NODE first, int item)
{
   NODE temp;
   temp = getnode();
   temp->info = item;
   temp->link = NULL;
   if (first == NULL)
      return temp;
   temp->link = first;
   first = temp;
   return first;
}
```

```
NODE insert_rear(NODE first, int item)
{
   NODE temp, cur;
   temp = getnode();
   temp->info = item;
   temp->link = NULL;
   if (first == NULL)
      return temp;
   cur = first;
   while (cur->link != NULL)
      cur = cur->link;
   cur->link = temp;
   return first;
}

NODE insert_pos(int item, int pos, NODE first)
{
   NODE temp;
   NODE prev, cur;
   int count;
   temp = getnode();
   temp->info = item;
   temp->link = NULL;
   if (first == NULL && pos == 1)
      return temp;
   if (first == NULL)
   {
      printf("invalid pos\n");
      return first;
   }
   if (pos == 1)
   {
      temp->link = first;
      return temp;
   }
   count = 1;
   prev = NULL;
   cur = first;
   while (cur != NULL && count != pos)
   {
      prev = cur;
      cur = cur->link;
      count++;
   }
```

```
    if (count == pos)
    {
        prev->link = temp;
        temp->link = cur;
        return first;
    }
    printf("IP\n");
    return first;
}
void display(NODE first)
{
    NODE temp;
    if (first == NULL)
        printf("list empty cannot display items\n");
else
        printf("Contents of the list:\n");
    for (temp = first; temp != NULL; temp = temp->link)
    {
        printf("%d\n", temp->info);
    }
}
void main()
{
    int item, choice, pos;
    NODE first = NULL;

    for (;;)
    {
        printf("\n1:Insert_front\n2:Insert_rear\n3:Insert_pos\n4:Display_list\n5:Exit\n");
        printf("Enter the choice\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter the item at front-end\n");
            scanf("%d", &item);
            first = insert_front(first, item);
            break;

        case 2:
            printf("Enter the item at rear-end\n");
            scanf("%d", &item);
            first = insert_rear(first, item);
            break;
```

```
        case 3:
            printf("Enter the position and item:\n");
            scanf("%d", &pos);
            scanf("%d",&item);
            first = insert_pos(item, pos, first);
            break;
        case 4:
            display(first);
            break;
        case 5:
            exit(0);
            break;
            default:printf("Invalid choice\n");

        }
    }
}
```

OUTPUT:

```
1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
4
list empty cannot display items

1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
1
Enter the item at front-end
1

1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
1
Enter the item at front-end
2

1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
2
Enter the item at rear-end
3
```

```
1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
4
Contents of the list:
2
1
3

1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
3
Enter the position and item:
2
8

1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
4
Contents of the list:
2
8
1
3
```

```
1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
8
Invalid choice

1:Insert_front
2:Insert_rear
3:Insert_pos
4:Display_list
5:Exit
Enter the choice
5

Process returned 0 (0x0)   execution time : 66.292 s
Press any key to continue.
```

2. WAP to Implement Singly Linked List with following operations

a) Create a linked list.
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list.

SOURCE CODE:

```
#include <stdio.h>
#include <conio.h>
struct node
{
   int info;
   struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
   NODE x;
   x = (NODE)malloc(sizeof(struct node));
   if (x == NULL)
   {
      printf("mem full\n");
      exit(0);
   }
   return x;
```

```
}
void freenode(NODE x)
{
   free(x);
}
NODE insert_front(NODE first, int item)
{
   NODE temp;
   temp = getnode();
   temp->info = item;
   temp->link = NULL;
   if (first == NULL)
      return temp;
   temp->link = first;
   first = temp;
   return first;
}
NODE delete_front(NODE first)
{
   NODE temp;
   if (first == NULL)
   {
      printf("List is empty cannot delete\n");
      return first;
   }
   temp = first;
   temp = temp->link;
   printf("Item deleted at front-end is=%d\n", first->info);
   free(first);
   return temp;
}
NODE insert_rear(NODE first, int item)
{
   NODE temp, cur;
   temp = getnode();
   temp->info = item;
   temp->link = NULL;
   if (first == NULL)
      return temp;
   cur = first;
   while (cur->link != NULL)
      cur = cur->link;
   cur->link = temp;
   return first;
```

```
}
NODE delete_rear(NODE first)
{
   NODE cur, prev;
   if (first == NULL)
   {
      printf("List is empty cannot delete\n");
      return first;
   }
   if (first->link == NULL)
   {
      printf("Item deleted is %d\n", first->info);
      free(first);
      return NULL;
   }
   prev = NULL;
   cur = first;
   while (cur->link != NULL)
   {
      prev = cur;
      cur = cur->link;
   }
   printf("Item deleted at rear-end is %d", cur->info);
   free(cur);
   prev->link = NULL;
   return first;
}

NODE delete_pos(int pos, NODE first)
{
   NODE prev, cur;
   int count;
   if (first == NULL || pos <= 0)
   {
      printf("Invalid position\n");
      return NULL;
   }
   if (pos == 1)
   {
      cur = first;
      first = first->link;
      printf("Item deleted is %d", cur->info);
      freenode(cur);
      return first;
```

```
    }
    prev = NULL;
    cur = first;
    count = 1;
    while (cur != NULL)
    {
        if (count == pos)
        {
            break;
        }
        prev = cur;
        cur = cur->link;
        count++;
    }
    if (count != pos)
    {
        printf("Invalid position\n");
        return first;
    }
    prev->link = cur->link;
    printf("Item deleted is %d", cur->info);
    freenode(cur);
    return first;
}
void display(NODE first)
{
    NODE temp;
    if (first == NULL)
        printf("List empty cannot display items\n");
else
        printf("Contents of the list:\n");
    for (temp = first; temp != NULL; temp = temp->link)
    {
        printf("%d\n", temp->info);
    }
}
void main()
{
    int item, choice, pos;
    NODE first = NULL;

    for (;;)
    {
```

```
        printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n 5:Delete_pos\n
6:Display_list\n 7:Exit\n");
        printf("Enter the choice\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
           printf("Enter the item at front-end\n");
           scanf("%d", &item);
           first = insert_front(first, item);
           break;
        case 2:
           first = delete_front(first);
           break;
        case 3:
           printf("Enter the item at rear-end\n");
           scanf("%d", &item);
           first = insert_rear(first, item);
           break;
        case 4:
           first = delete_rear(first);
           break;
        case 5:
           printf("Enter the position:\n");
           scanf("%d", &pos);
           first = delete_pos(pos, first);
           break;
        case 6:
           display(first);
           break;
        case 7:
           exit(0);
           break;
           default:printf("Invalid choice\n");
        }
     }
}
```

OUTPUT:

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
5
Enter the position:
7
Invalid position

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
6
List empty cannot display items

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
1
Enter the item at front-end
3
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
1
Enter the item at front-end
5

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
3
Enter the item at rear-end
7
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
6
Contents of the list:
5
3
7

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
5
Enter the position:
2
Item deleted is 3
```

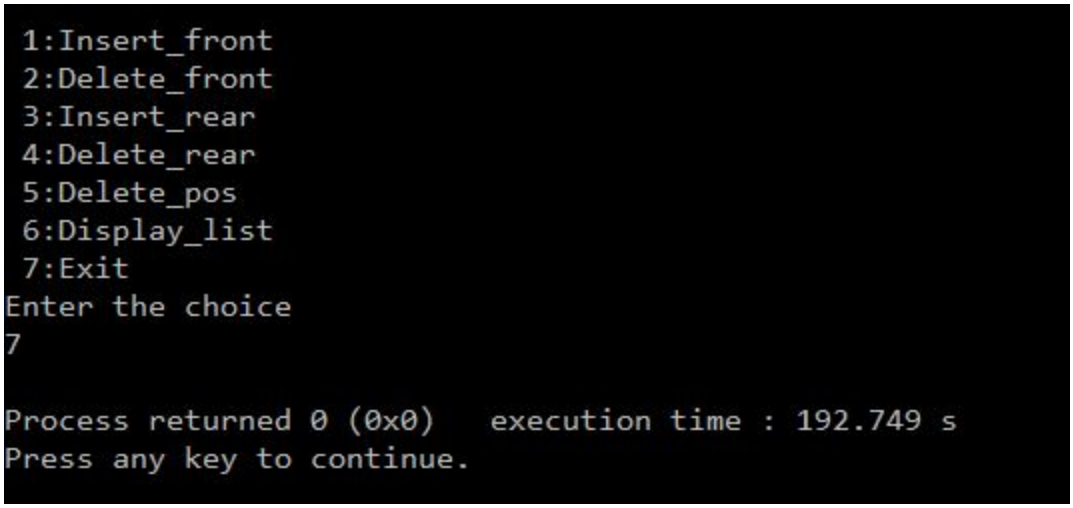
```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
6
Contents of the list:
5
7

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
2
Item deleted at front-end is=5
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
4
Item deleted is 7

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
2
List is empty cannot delete

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete_pos
 6:Display_list
 7:Exit
Enter the choice
9
Invalid choice
```



3. WAP Implement Single Link List with following operations

a) Sort the linked list.

b) Reverse the linked list.

c) Concatenation of two linked lists

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
  int info;
  struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE)malloc(sizeof(struct node));
if(x==NULL)
 {
  printf("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
free(x);
}
```

```
NODE insert_front(NODE first,int item)
{
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
NODE delete_front(NODE first)
{
NODE temp;
if(first==NULL)
{
printf("List is empty cannot delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("Item deleted at front-end is=%d\n",first->info);
free(first);
return temp;
}
NODE insert_rear(NODE first,int item)
{
NODE temp,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
 return temp;
cur=first;
while(cur->link!=NULL)
 cur=cur->link;
cur->link=temp;
return first;
}
NODE delete_rear(NODE first)
{
NODE cur,prev;
if(first==NULL)
```

```
{
printf("List is empty cannot delete\n");
return first;
}
if(first->link==NULL)
{
printf("Item deleted is %d\n",first->info);
free(first);
return NULL;
}
prev=NULL;
cur=first;
while(cur->link!=NULL)
{
prev=cur;
cur=cur->link;
}
printf("Item deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;
}
NODE order_list(int item,NODE first)
{
NODE temp,prev,cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL) return temp;
if(item<first->info)
{
temp->link=first;
return temp;
}
prev=NULL;
cur=first;
while(cur!=NULL&&item>cur->info)
{
prev=cur;
cur=cur->link;
}
prev->link=temp;
temp->link=cur;
return first;
```

```
}

void display(NODE first)
{
 NODE temp;
 if(first==NULL)
 printf("List empty cannot display items\n");
else
 printf("Contents of the list:\n");
 for(temp=first;temp!=NULL;temp=temp->link)
  {
  printf("%d\n",temp->info);
  }
}
NODE concat(NODE first,NODE second)
{
 NODE cur;
 if(first==NULL)
  return second;
 if(second==NULL)
  return first;
 cur=first;
 while(cur->link!=NULL)
  cur=cur->link;
 cur->link=second;
 return first;
}
NODE reverse(NODE first)
 {
 NODE cur,temp;
 cur=NULL;
 while(first!=NULL)
  {
   temp=first;
   first=first->link;
   temp->link=cur;
   cur=temp;
  }
 return cur;
}

void main()
{
int item,choice,key,n,i;
```

```
NODE first=NULL,a,b;
for(;;)
{
printf("\n1:Insert_front\n2:Delete_front\n3:Insert_rear\n4:Delete_rear\n");
printf("5:Order_list\n6:Display_list\n7:Concat\n8:Reverse\n9:Exit\n");
printf("Enter the choice\n");
scanf("%d",&choice);
switch(choice)
 {
  case 1:printf("Enter the item at front-end\n");
           scanf("%d",&item);
           first=insert_front(first,item);
           break;
  case 2:first=delete_front(first);
           break;
  case 3:printf("Enter the item at rear-end\n");
           scanf("%d",&item);
           first=insert_rear(first,item);
           break;
  case 4:first=delete_rear(first);
           break;
  case 5:printf("Enter the item to be inserted in ordered_list\n");
           scanf("%d",&item);
           first=order_list(item,first);
           break;
  case 6:display(first);
           break;
  case 7:printf("Enter the no of nodes in 1\n");
                    scanf("%d",&n);
                    a=NULL;
                    for(i=0;i<n;i++)
                     {
                      printf("Enter the item\n");
                      scanf("%d",&item);
                      a=insert_rear(a,item);
                     }
                     printf("Enter the no of nodes in 2\n");
                    scanf("%d",&n);
                    b=NULL;
                    for(i=0;i<n;i++)
                     {
                      printf("Enter the item\n");
                      scanf("%d",&item);
                      b=insert_rear(b,item);
```

```
                    }
                    a=concat(a,b);
                    display(a);
                   break;
  case 8:first=reverse(first);
                   display(first);
                   break;
 case 9:exit(0);
         break;
         default:printf("Invalid choice\n");
 }
}

}
```

OUTPUT:

```
1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
8
List empty cannot display items

1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
5
Enter the item to be inserted in ordered_list
8
```

```
1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
5
Enter the item to be inserted in ordered_list
2

1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
5
Enter the item to be inserted in ordered_list
5
```

```
1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
6
Contents of the list:
2
5
8

1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
8
Contents of the list:
8
5
2
```

```
1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
7
Enter the no of nodes in 1
2
Enter the item
1
Enter the item
2
Enter the no of nodes in 2
3
Enter the item
3
Enter the item
4
Enter the item
5
Contents of the list:
1
2
3
4
5
```

```
1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
10
Invalid choice

1:Insert_front
2:Delete_front
3:Insert_rear
4:Delete_rear
5:Order_list
6:Display_list
7:Concat
8:Reverse
9:Exit
Enter the choice
9

Process returned 0 (0x0)   execution time : 78.062 s
Press any key to continue.
```