Linked LIST

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

struct node{
   int data;
   struct node *next;

};
void display(struct node *d_head)
{
   while(d_head != NULL)
   {
      printf("%d ",d_head->data);
      d_head=d_head->next;
   }
   printf("\n");
}

struct node *create(int arr[], int size){

   struct node *head=NULL;
   struct node *temp=NULL;
```

```
    struct node *current=NULL;
    for(int i=0; i<size; i++)
    {
        temp = (struct node*)malloc(sizeof(struct node));
        temp->data = arr[i];
        temp->next = NULL;
        if(head==NULL)
        {
            head = temp;
            current = temp;
        }
        else
        {
            current->next = temp;
            current = current->next;
        }
    }
    return(head);
};
struct node *del(struct node *head, int item){
    struct node *dummyhead=NULL;
    dummyhead = (struct node*)malloc(sizeof(struct node));
    dummyhead->next=head;
```

```
    struct node *temp=dummyhead;

    while(temp->next != NULL)
    {
        if(temp->next->data == item )
        {
            temp->next=temp->next->next;
        }
        temp=temp->next;
    }
    return(dummyhead->next);
};

void insert_begin(struct node *head, int n_data)
{
    struct node *temp=NULL;
    temp = (struct node*)malloc(sizeof(struct node*));

    temp->data = n_data;
    temp->next = head;

    struct node *n_head;
    n_head = temp;
```

```
    display(n_head);
}
void insert_mid(struct node *head,int position, int n_data)
{
    struct node *temp=head;
    int count =0;
    while(temp != NULL)
    {
        count++;
        if(count==position)
        {
            struct node *newnode=NULL;
            newnode = (struct node *)malloc(sizeof(struct node));

            newnode->data=n_data;
            newnode->next=temp->next;
            temp->next=newnode;
        }
        temp=temp->next;
    }

    display(head);
```

```
}
void insert_end(struct node *head, int n_data)
{
   struct node *current=head;
   while(current->next != NULL)
   {
      current = current->next;
   }

   struct node *temp=NULL;
   temp = (struct node*)malloc(sizeof(struct node*));
   temp->data = n_data;
   temp->next = NULL;
   current->next = temp;

   display(head);
}

int main()
{
   int arr[5]={10,20,30,40,50};
   struct node *head=NULL;
```

```
    head = create(arr,5);


    display(head);


    insert_begin(head, 5);
    insert_end(head, 55);
    insert_mid(head,2, 100);
    struct node *afterdelete_head=del(head,20);
    display(afterdelete_head);
}
```

TREE

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


struct node{


    int data;
    struct node *left;
    struct node *right;


};
struct node *createnode(int item)
```

```
{
   struct node *n_node;
   n_node = (struct node *)malloc(sizeof(struct node));
   n_node->data = item;
   n_node->left = NULL;
   n_node->right = NULL;

   return(n_node);

};

void pre_display(struct node *d_root)
{
   if(d_root==NULL)
   {
      return;
   }
   printf("%d ",d_root->data);
   pre_display(d_root->left);
   pre_display(d_root->right);

}
void in_display(struct node *d_root)
```

```
{
   if(d_root==NULL)
   {
      return;
   }

   in_display(d_root->left);
   printf("%d ",d_root->data);
   in_display(d_root->right);

}
void post_display(struct node *d_root)
{
   if(d_root==NULL)
   {
      return;
   }

   post_display(d_root->left);
   post_display(d_root->right);
   printf("%d ",d_root->data);

}
```

```
/*
                1
               / \
              2   3
             / \ / \
            4  5 6  7

*/

int main()
{
   struct node *root;

   root = createnode(1);
   root->left = createnode(2);
   root->right = createnode(3);
   root->left->left = createnode(4);
   root->left->right = createnode(5);
   root->right->left = createnode(6);
   root->right->right = createnode(7);

   printf("Preorder: ");
```

```
    pre_display(root);


    printf("\nInorder: ");
    in_display(root);


    printf("\nPostorder: ");
    post_display(root);



}
```

BUBBLE SORT

```
#include<stdio.h>
int main()
{
    int arr[5]= {3,7,1,9,4};
    int size=5;
    int temp;
    for(int i=0; i<size-1; i++)
    {
        int swap=0;
        for(int j=0; j<size-1-i; j++)
        {
            if(arr[j]>arr[j+1])
```

```
        {
            temp=arr[j];
            arr[j]=arr[j+1];
            arr[j+1]=temp;
            swap=1;
        }
    }
    if(swap==0)
    {
        break;
    }
  }
  for(int j=0; j<5; j++)
  {
    printf("%d ",arr[j]);
  }
}
```

Binary Search

```
#include<stdio.h>
int main()
{
  int arr[10]= {1,2,4,5,7,9,10,23,56,78};
  int item=99;
```

```
int size=10;
int left=0;
int right=size-1;
int middle=(left+right)/2;

while(left<=right)
{
   if(arr[middle]==item)
   {
      printf("Item index: %d\n",middle);
      return 0;
   }
   else if(arr[middle]>item)
   {
      right=middle-1;
      middle=(left+right)/2;
   }
   else if(arr[middle]<item)
   {
      left=middle+1;
      middle=(left+right)/2;
   }
```

```
    }
    printf("Item not found\n");
    return 0;
}
```

Linear Search

```
#include<stdio.h>
int main()
{
    int arr[5]={4,6,8,3,5};
    int search=10;

    for(int i=0; i<5; i++)
    {
        if(arr[i]==search)
        {
            printf("Item found at index: %d, Item is:%d",i,search);
            return 0;
        }
    }
    printf("Item not found\n");
}
```

Matrix Multiplication

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

```
int main()
{
   int r1,r2,c1,c2;

   printf("Enter the value of Row-1, Column-1, Row-2, Column-2\n");
   scanf("%d %d %d %d",&r1,&c1,&r2,&c2);
   while( c1 != r2)
   {
      printf(" Column-1 and Row-2 is not equal\n");
      printf("Enter the value of Row-1, Column-1, Row-2, Column-2\n");
      scanf("%d %d %d %d",&r1,&c1,&r2,&c2);

   }

   int mat1[r1][c1];
   int mat2[r2][c2];
   int mult[r1][c2];

   for(int i=0; i<r1; i++)
   {
      for(int j=0; j<c1; j++)
      {
         printf("Enter the element of [%d][%d]=",i,j);
```

```
        scanf("%d",&mat1[i][j]);
    }
}
printf("\n");

for(int i=0; i<r2; i++)
{
    for(int j=0; j<c2; j++)
    {
        printf("Enter the element of [%d][%d]=",i,j);
        scanf("%d",&mat2[i][j]);
    }
}
printf("\n");

printf("First Matrix:\n");
for(int i=0; i<r1; i++)
{
    for(int j=0; j<c1; j++)
    {
        printf("%d ",mat1[i][j]);
    }
    printf("\n");
```

```
}

printf("Second Matrix:\n");
for(int i=0; i<r2; i++)
{
   for(int j=0; j<c2; j++)
   {
      printf("%d ",mat2[i][j]);
   }
   printf("\n");
}

printf("\n");
int sum=0;
for(int i=0; i<r1; i++)
{
   for(int j=0; j<c2; j++)
   {
      for(int k=0; k<c1/*r2*/; k++)
      {
         sum = sum + (mat1[i][k] * mat2[k][j]);
      }
      mult[i][j]=sum;
```

```
            sum=0;
        }
    }

    printf("Multiplied Matrix:\n");
    for(int i=0; i<r1; i++)
    {
        for(int j=0; j<c2; j++)
        {
            printf("%d ",mult[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Array Insert

```
#include<stdio.h>
int main()
{
    int array_size;
    printf("Enter the size of the array: ");
```

```
scanf("%d",&array_size);

int arr[array_size];

int n;
printf("\nNumber of elements in the array: ");
scanf("%d",&n);

for(int i=0; i<n; i++)
{
   printf("Enter the elements of position [%d]=",i);
   scanf("%d",&arr[i]);
}

int value;
int position;
printf("\nWhich value you want to insert: ");
scanf("%d",&value);
printf("\nWhich position you want to insert: ");
scanf("%d",&position);

if(n==array_size)
{
```

```
    printf("Array is full\n");
}
else if(position > array_size-1 || position < 0)
{
    printf("Position is not valid\n");
}
else
{
    for(int i=n; i>=position; i--)
    {
        arr[i]=arr[i-1];
    }
    arr[position-1]=value;

    printf("\nFinal Array\n");
    for(int i=0; i<=n; i++)
    {

        printf("%d ",arr[i]);
    }
}

return 0;
```

```
}
```