Linked list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
struct node {
  int value;
  struct node *next;
};
void insert();
void display();
typedef struct node DATA_NODE;
DATA_NODE    *head_node,    *first_node,
*temp_node = 0, *prev_node, next_node;
int data;
int main() {
  int option = 0;
  printf("Singly Linked List: \n");
  while (option < 3) {
    printf("\nOptions\n");
    printf("1 : Insert into Linked List \n");
    printf("2 : Delete from Linked List \n");
    printf("Others : Exit()\n");
    printf("Enter your option:");
    scanf("%d", &option);
    switch (option) {
      case 1:
        insert();
        break;
      case 2:
        delete();
        break;
      default:
        break;
    }
  }
  return 0;
}
void insert() {
   printf("\nEnter Element for Insert Linked
List : \n");
  scanf("%d", &data);
      temp_node    =    (DATA_NODE    *)
malloc(sizeof (DATA_NODE));
  temp_node->value = data;
  if (first_node == 0) {
    first_node = temp_node;
  } else {
    head_node->next = temp_node;
  }
  temp_node->next = 0;
  head_node = temp_node;
  fflush(stdin);
}
void delete() {
  int countvalue, pos, i = 0;
  countvalue = count();
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");
  printf("\nEnter Position for Delete Element :
\n");
  scanf("%d", &pos);
  if (pos > 0 && pos <= countvalue) {
    if (pos == 1) {
      temp_node = temp_node -> next;
      first_node = temp_node;
      printf("\nDeleted Successfully \n\n");
    } else {
      while (temp_node != 0) {
        if (i == (pos - 1)) {
          prev_node->next = temp_node->next;
          if(i == (countvalue - 1))
          {
     head_node = prev_node;
    }
          printf("\nDeleted Successfully \n\n");
          break;
        } else {
          i++;
          prev_node = temp_node;
          temp_node = temp_node -> next;
        }
      }
    }
  } else
    printf("\nInvalid Position \n\n");
}
```

Quick Sort

```
#include<stdio.h>
void  quicksort(int  number[25],int  first,int
last){
  int i, j, pivot, temp;
  if(first<last){
    pivot=first;
    i=first;
    j=last;
    while(i<j){

while(number[i]<=number[pivot]&&i<last)
      i++;
      while(number[j]>number[pivot])
      j--;
      if(i<j){
        temp=number[i];
        number[i]=number[j];
        number[j]=temp;
      }
    }
    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);
  }
}
int main(){
  int i, count, number[25];
  printf("How many elements are u going to
enter?: ");
  scanf("%d",&count);
  printf("Enter %d elements: ", count);
  for(i=0;i<count;i++)
  scanf("%d",&number[i]);
  quicksort(number,0,count-1);
  printf("Order of Sorted elements: ");
  for(i=0;i<count;i++)
  printf(" %d",number[i]);
  return 0;
}
```

Bubble sort

```
#include <stdio.h>

int main()

{

  int array[100], n, c, d, swap;

  printf("Enter number of elements\n");

  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)

    scanf("%d", &array[c]);

  for (c = 0 ; c < n - 1; c++)

  {

    for (d = 0 ; d < n - c - 1; d++)

    {

      if (array[d] > array[d+1]) /* For
decreasing order use '<' instead of '>' */

      {

        swap       = array[d];

        array[d]   = array[d+1];

        array[d+1] = swap;

      }

    }

  }

  printf("Sorted list in ascending order:\n");

  for (c = 0; c < n; c++)

     printf("%d\n", array[c]);

  return 0;

}
```

Binary tree

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

int value;

struct node *left_child, *right_child;

};

struct node *new_node(int value)

{

struct node *tmp = (struct node *)malloc(sizeof(struct node));

tmp->value = value;

tmp->left_child = tmp->right_child = NULL;

return tmp;

}

void print(struct node *root_node) // displaying the nodes!

{

if (root_node != NULL)

{

print(root_node->left_child);

printf("%d \n", root_node->value);

print(root_node->right_child);

}

}

struct node* insert_node(struct node* node, int value) // inserting nodes!

{

if (node == NULL) return new_node(value);

if (value < node->value)

{

node->left_child = insert_node(node->left_child, value);

}

else if (value > node->value)

{

node->right_child = insert_node(node->right_child, value);

}

return node;

}

int main()

{

printf("Implementation of a Binary Tree in C! \n\n");

struct node *root_node = NULL;

root_node = insert_node(root_node, 10);

insert_node(root_node, 10);

insert_node(root_node, 30);

insert_node(root_node, 25);

insert_node(root_node, 36);

insert_node(root_node, 56);

insert_node(root_node, 78);

print(root_node);

return 0;

}
```

Heap sort

```
#include <stdio.h>
void main()
{
    int heap[10], no, i, j, c, root, temp;
    printf("\n Enter no of elements :");
    scanf("%d", &no);
    printf("\n Enter the nos : ");
    for (i = 0; i < no; i++)
       scanf("%d", &heap[i]);
    for (i = 1; i < no; i++)
    {
        c = i;
        do
        {
            root = (c - 1) / 2;
             if (heap[root] < heap[c])   /* to create
MAX heap array */
            {
                temp = heap[root];
                heap[root] = heap[c];
                heap[c] = temp;
            }
            c = root;
        } while (c != 0);
    }
    printf("Heap array : ");
    for (i = 0; i < no; i++)
        printf("%d\t ", heap[i]);
    for (j = no - 1; j >= 0; j--)
    {
        temp = heap[0];
        heap[0] = heap[j    /* swap max element
with rightmost leaf element */
        heap[j] = temp;
        root = 0;
        do
        {
             c = 2 * root + 1;    /* left node of root
element */
            if ((heap[c] < heap[c + 1]) && c < j-1)
                c++;
               if  (heap[root]<heap[c]  &&  c<j)     /*
again rearrange to max heap array */
            {
                temp = heap[root];
                heap[root] = heap[c];
                heap[c] = temp;
            }
            root = c;
        } while (c < j);
    }
    printf("\n The sorted array is : ");
    for (i = 0; i < no; i++)
       printf("\t %d", heap[i]);
    printf("\n Complexity : \n Best case = Avg
case = Worst case = O(n logn) \n");
}
```

Selection sort

```
#include <stdio.h>

int main() {

  int arr[10]={6,12,0,18,11,99,55,45,34,2};

  int n=10;

  int i, j, position, swap;

  for (i = 0; i < (n - 1); i++) {

    position = i;

    for (j = i + 1; j < n; j++) {

      if (arr[position] > arr[j])

        position = j;

    }

    if (position != i) {

      swap = arr[i];

      arr[i] = arr[position];

      arr[position] = swap;

    }

  }

  for (i = 0; i < n; i++)

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

  return 0;

}
```

Insertion sort

```
#include <stdio.h>

int main()

{

   int n, i, j, temp;

   int arr[64];

   printf("Enter number of elements\n");

   scanf("%d", &n);

   printf("Enter %d integers\n", n);

   for (i = 0; i < n; i++)

   {

      scanf("%d", &arr[i]);

   }

   for (i = 1 ; i <= n - 1; i++)

   {

    j = i;

         while ( j > 0 && arr[j-1] > arr[j])

         {

            temp     = arr[j];

            arr[j]   = arr[j-1];

            arr[j-1] = temp;

            j--;

         }

   }

   printf("Sorted list in ascending order:\n");

   for (i = 0; i <= n - 1; i++)

   {

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

   }

   return 0;

}
```

Merge sort

```
#include<stdio.h>
void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
int main()
{
int a[30],n,i;
printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter array elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
mergesort(a,0,n-1);
printf("\nSorted array is :");
for(i=0;i<n;i++)
printf("%d ",a[i]);
return 0;
}
void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
mid=(i+j)/2;
mergesort(a,i,mid); //left recursion
mergesort(a,mid+1,j); //right recursion
merge(a,i,mid,mid+1,j); //merging of two
sorted sub-arrays
}
}
void merge(int a[],int i1,int j1,int i2,int j2)
{
int temp[50]; //array used for merging
int i,j,k;
i=i1; //beginning of the first list
j=i2; //beginning of the second list
k=0;
while(i<=j1 && j<=j2) //while elements in
both lists
{
if(a[i]<a[j])
temp[k++]=a[i++];
else
temp[k++]=a[j++];
}
while(i<=j1) //copy remaining elements of
the first list
temp[k++]=a[i++];
while(j<=j2) //copy remaining elements of
the second list
temp[k++]=a[j++];
//Transfer elements from temp[] back to a[]
for(i=i1,j=0;i<=j2;i++,j++)
a[i]=temp[j];
}
```

Linear search

```
#include <stdio.h>

int main()

{

 int array[100], search, c, n;

  printf("Enter number of elements in
array\n");

 scanf("%d", &n);

 printf("Enter %d integer(s)\n", n);

 for (c = 0; c < n; c++)

  scanf("%d", &array[c]);

 printf("Enter a number to search\n");

 scanf("%d", &search);

 for (c = 0; c < n; c++)

 {

   if (array[c] == search)    /* If required
element is found */

   {

     printf("%d is present at location %d.\n",
search, c+1);

     break;

   }

 }

 if (c == n)

   printf("%d isn't present in the array.\n",
search);

 return 0;

}
```

Binary search

```
#include <stdio.h>

int main()

{

   int  c,  first,  last,  middle,  n,  search,
array[100];

  printf("Enter number of elements\n");

  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++)

    scanf("%d", &array[c]);

  printf("Enter value to find\n");

  scanf("%d", &search);

  first = 0;

  last = n - 1;

  middle = (first+last)/2;

  while (first <= last) {

    if (array[middle] < search)

      first = middle + 1;

    else if (array[middle] == search) {

         printf("%d  found  at  location  %d.\n",
search, middle+1);

      break;

    }

    else

      last = middle - 1;

    middle = (first + last)/2;

  }

  if (first > last)

     printf("Not found! %d isn't present in the
list.\n", search);

  return 0;

}
```

Created with Mi Notes

Binary search tree

```
#include<stdio.h>
#include<stdlib.h>
typedef struct BST
{
int data;
struct BST *left;
struct BST *right;
}node;
node *create();
void insert(node *,node *);
void preorder(node *);
int main()
{
char ch;
node *root=NULL,*temp;
do
{
temp=create();
if(root==NULL)
root=temp;
else
insert(root,temp);
printf("nDo you want to enter more(y/n)?");
getchar();
scanf("%c",&ch);
}while(ch=='y'|ch=='Y');
printf("nPreorder Traversal: ");
preorder(root);
return 0;
}
node *create()
{
node *temp;
printf("nEnter data:");
temp=(node*)malloc(sizeof(node));
scanf("%d",&temp->data);
temp->left=temp->right=NULL;
return temp;
}
void insert(node *root,node *temp)
{
if(temp->data<root->data)
{
if(root->left!=NULL)
insert(root->left,temp);
else
root->left=temp;
}
if(temp->data>root->data)
{
if(root->right!=NULL)
insert(root->right,temp);
else
root->right=temp;
}
}
void preorder(node *root)
{
if(root!=NULL)
{
printf("%d ",root->data);
preorder(root->left);
preorder(root->right);
}
}
```

Stack

```
#include <stdio.h>
#define MAXSIZE 5
struct stack
{
    int stk[MAXSIZE];
    int top;
};
typedef struct stack STACK;
STACK s;
void push(void);
int  pop(void);
void display(void);
void main ()
{
    int choice;
    int option = 1;
    s.top = -1;
    printf ("STACK OPERATION\n");
    while (option)
    {
        printf ("------------------------------------------\n");
        printf ("      1    -->    PUSH               \n");
        printf ("      2    -->    POP               \n");
         printf ("      3    -->    DISPLAY
\n");
        printf ("      4    -->    EXIT           \n");
        printf ("------------------------------------------\n");
        printf ("Enter your choice\n");
        scanf    ("%d", &choice);
        switch (choice)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            return;
        }
        fflush (stdin);
        printf ("Do you want to continue(Type 0
or 1)?\n");
        scanf    ("%d", &option);
    }
}
/*  Function to add an element to the stack
*/
void push ()
{
    int num;
    if (s.top == (MAXSIZE - 1))
    {
        printf ("Stack is Full\n");
        return;
    }
    else
    {
            printf  ("Enter  the  element  to  be
pushed\n");
        scanf ("%d", &num);
        s.top = s.top + 1;
        s.stk[s.top] = num;
    }
    return;
}
/*  Function to delete an element from the
stack */
int pop ()
{
    int num;
    if (s.top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s.top);
    }
    else
    {
        num = s.stk[s.top];
             printf  ("poped  element  is  =  %dn",
s.stk[s.top]);
        s.top = s.top - 1;
    }
    return(num);
}
/*   Function  to  display  the  status  of  the
stack */
void display ()
{
    int i;
    if (s.top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
         printf ("\n The status of the stack is
\n");
        for (i = s.top; i >= 0; i--)
        {
            printf ("%d\n", s.stk[i]);
        }
    }
    printf ("\n");
}
```

Queue

```
#include <stdio.h>
#define MAX 50
void insert();
void delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
void main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */
void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
} /* End of insert() */
void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */
void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
} /* End of display() */
```

Matrix multiplication

```
#include <stdio.h>

int main()

{

  int m, n, p, q, c, d, k, sum = 0;

     int first[10][10], second[10][10],
multiply[10][10];

   printf("Enter the number of rows and
columns of first matrix\n");

  scanf("%d%d", &m, &n);

    printf("Enter the elements of first
matrix\n");

  for (  c = 0 ; c < m ; c++ )

    for ( d = 0 ; d < n ; d++ )

      scanf("%d", &first[c][d]);

   printf("Enter the number of rows and
columns of second matrix\n");

  scanf("%d%d", &p, &q);

  if ( n != p )

    printf("Matrices with entered orders can't
be multiplied with each other.\n");

  else

  {

      printf("Enter the elements of second
matrix\n");

    for ( c = 0 ; c < p ; c++ )

      for ( d = 0 ; d < q ; d++ )

        scanf("%d", &second[c][d]);

    for ( c = 0 ; c < m ; c++ )

    {

      for ( d = 0 ; d < q ; d++ )

      {

        for ( k = 0 ; k < p ; k++ )

        {

          sum = sum + first[c][k]*second[k][d];

        }

        multiply[c][d] = sum;

        sum = 0;

      }

    }

    printf("Product of entered matrices:-\n");

    for ( c = 0 ; c < m ; c++ )

    {

      for ( d = 0 ; d < q ; d++ )

        printf("%d\t", multiply[c][d]);

      printf("\n");

    }

  }

  return 0;

}
```

Matrix sum and difference

```
#include <stdio.h>
void main()
{
int     array1[10][10],     array2[10][10],
arraysum[10][10], arraydiff[10][10];
int i, j, m, n, option;
printf("Enter the order of the matrix array1
and array2 \n");
scanf("%d %d", &m, &n);
printf("Enter the elements of matrix array1
\n");
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
scanf("%d", &array1[i][j]);
}
}
printf("MATRIX array1 is \n");
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
printf("%3d", array1[i][j]);
}
printf("\n");
}
printf("Enter the elements of matrix array2
\n");
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
scanf("%d", &array2[i][j]);
}
}
printf("MATRIX array2 is \n");
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
printf("%3d", array2[i][j]);
}
printf("\n");
}
printf("Enter your option: 1 for Addition and
2 for Subtraction \n");
scanf("%d", &option);
switch (option)
{
case 1:
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
arraysum[i][j] = array1[i][j] + array2[i][j];
}
}
printf("Sum matrix is \n");
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
printf("%3d", arraysum[i][j]) ;
}
printf("\n");
}
break;
case 2:
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
arraydiff[i][j] = array1[i][j] - array2[i][j];
}
}
printf("Difference matrix is \n");
for (i = 0; i < m; i++)
{
for (j = 0; j < n; j++)
{
printf("%3d", arraydiff[i][j]) ;
}
printf("\n");
}
break;
}
}
```