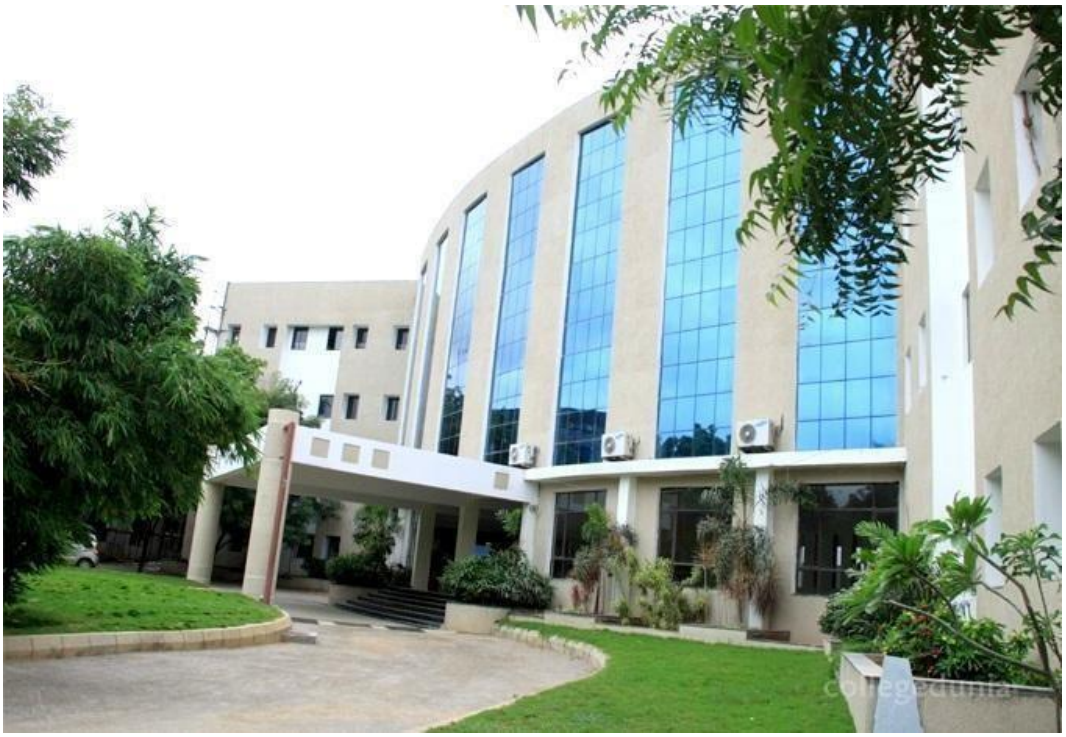
## DEPARTMENT OF COMPUTER SCIENCE ENGINEERING – DATA SCIENCE



# DATA STRUCTURES

# LAB MANUAL

Subject Code : KG23ACD218

Regulation : R18/JNTUH

Academic Year : 2019-2020

## II B. TECH I SEMESTER

**COMPUTER SCIENCE AND ENGINEERING – DATA SCIENCE**

**KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY**

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING – DATA SCIENCE

# VISION AND MISSION OF THE INSTITUTION

### VISION

To become self-sustainable institution this is recognized for its new age engineering through innovative teaching and learning culture, inculcating research and entrepreneurial ecosystem, and sustainable social impact in the community.

### MISSION

- To offer undergraduate and post-graduate programs that is supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
- To provide necessary support structures for students, this will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
- To provide faculty and students with an ecosystem that fosters research and development through strategic partnerships with government organisations and collaboration with industries.
- To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic growth.

# VISION AND MISSION OF DEPARTMENT

### VISION

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

### MISSION

- To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills while working with the community to address societal issues.

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING – DATA SCIENCE

## PROGRAM EDUCATIONAL OBJECTIVES (PEOS):

A graduate of the Computer Science and Engineering Program should:

| | |
|---|---|
| PEO1 | **Program Educational Objective1: (PEO1)**<br>The Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles. |
| PEO2 | **Program Educational Objective2 :( PEO2)**<br>**The** Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees. |
| PEO3 | **Program Educational Objective3: (PEO3)**<br>The Graduates will communicate effectively, work collaboratively and exhibit high levels of Professionalism, moral and ethical responsibility. |
| PEO4 | **Program Educational Objective4 :( PEO4)**<br>**The** Graduates will develop the ability to understand and analyse Engineering issues in a broader perspective with ethical responsibility towards sustainable development. |

## PROGRAM OUTCOMES (POS):

| | |
|---|---|
| PO1 | **Engineeringknowledge**:Applytheknowledgeofmathematics,science,engineering Fundamentals andanengineeringspecializationtothesolutionofcomplexengineeringproblems. |
| PO2 | **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. |
| PO3 | **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations. |
| PO4 | **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions. |

| | |
|---|---|
| PO5 | **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |
| PO6 | **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice. |
| PO7 | **Environment and sustainability**: Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| PO8 | **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice. |
| PO9 | **Individual and team work**: Function effectively as an individual, and as a member or leader In diverse teams, and in multi-disciplinary settings. |
| PO10 | **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions. |
| PO11 | **Project management and finance**: Demonstrate knowledge and understanding of the Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments. |
| PO12 | **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change. |

## PROGRAM SPECIFIC OUTCOMES(PSOS):

| | |
|---|---|
| PSO1 | **Problem Solving Skills** – Graduate will be able to apply computational techniques and software principles to solve complex engineering problems pertaining to software engineering. |
| PSO2 | **Professional Skills** – Graduate will be able to think critically, communicate effectively, and collaborate in teams through participation in co and extra-curricular activities. |
| PSO3 | **Successful Career** – Graduates will possess a solid foundation in computer science and engineering that will enable them to grow in their profession and pursue lifelong learning through post-graduation and professional development. |

## PREREQUISITES:

Acourseon "ProgrammingforProblemSolving"

.

## COURSE OBJECTIVES:

- ItcoversvariousconceptsofCprogramminglanguage
- Itintroducessearchingand sorting algorithms
- It providesanunderstandingofdatastructuressuchasstacksandqueues.

## COURSE OUTCOME

- Ability to develop C programs for computing and real-life applications using basicelementslike control statements, arrays, functions, pointers and strings, and data structures like stacks,queuesand linkedlists.
- AbilitytoImplementsearchingandsortingalgorithms

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING – DATA SCIENCE

**Course Name: DS LAB** **Course Code:KG23ACD218**

**Year/Semester: II/I** **Regulation: KG23**

**ListofExperiments**

1. Writeaprogramthatusesfunctionstoperformthefollowingoperationsonsinglylinkedlist.:
   i) Creation ii)Insertion iii)Deletion iv)Traversal
2. Writeaprogramthatusesfunctionstoperformthefollowingoperationsondoublylinkedlist.:
   i) Creation ii)Insertion iii)Deletion iv)Traversal
3. Writeaprogramthatusesfunctionstoperformthefollowingoperationsoncircularlinkedlist.:
   i) Creation ii)Insertion iii)Deletion iv)Traversal
4. Writeaprogram thatimplementstack(itsoperations)using
   i) Arrays ii)Pointers
5. WriteaprogramthatimplementQueue(itsoperations)using
   i) Arrays ii)Pointers
6. Writeaprogramthatimplements the followingsortingmethods tosortagiven listofintegersin ascending order
   i) Bubblesort ii)Selectionsort iii)Insertionsort
7. Writeaprogramthatusebothrecursiveandnonrecursive functions toperformthefollowingsearching operationsfora Keyvalueina given listofintegers:
   i) Linearsearch ii)Binarysearch
8. Writeaprogramtoimplementthetreetraversalmethods.
9. Writeaprogram toimplementthegraphtraversalmethods.




**FACULTY** **HOD, CSE**

## Week: 1

**Write a C program to perform various operations such as creation, insertion, deletion, search and display on single linked list**

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
void create();
void insert();
void del();
void display();
struct node
{
 int data;
 struct node *link;
};
struct node *first = null, *last = null ,*next, *curr, *prev;
int ch;
void main()
{
clrscr();
printf("singly linked list \n");
 do
 {
printf("\n 1.create \n 2.insert \n 3.delete \n 4.exit \n ");
printf("Enter your choice");
scanf("%d", &ch);
  switch(ch)
  {
   case 1: create();
           display();
           break;
   case 2: insert();
           display();
           break;
   case 3: del();
           display();
           break;
   case 4: exit(0);
  }
 }
 while(ch<=3);
}
void create()
{
curr = (struct node *) malloc(sizeof(struct node));
```

```
printf("Enter the data: ");
scanf("%d", &curr -> data);
curr -> link = null;
 first = curr;
 last = curr;
}
void insert()
{
 int pos, c = 1;
curr=(struct node *)malloc(sizeof(struct node));
printf("Enter the data:");
scanf("%d", &curr -> data);
printf("Enter the position:");
scanf("%d", &pos);
 if((pos == 1) && (first != null))
 {
curr -> link = first;
  first = curr;
 }
 else
 {
  next = first;
  while(c < pos)
  {
prev = next;
   next = prev -> link;
c++;
  }
  if(prev == null)
  {
printf("\n Invalid position");
 }
 else
 {
curr -> link = prev -> link;
prev -> link = curr;
  if(curr -> link == null)
  {
   last = curr;
  }
 }
}
void del()
{
 int pos, c = 1;
printf("Enter the position");
scanf("%d", &pos);
 if(first = null)
 {
```

```
printf("\n list is empty");
 }
 else if(pos == 1) && (first -> link == null)
 {
printf("\n Deleted element is %d \n", curr -> data);
  free(curr);
 }
 else
 {
  next = first;
  while(c < pos)
  {
prev = next;
   next = next -> link;
c++;
  }
prev -> link = next -> link;
  next -> link = null;
  if(next = null)
  {
printf("\n Invalid position");
  }
  else
  {
printf("\n Deleted element is:%d\n", next -> data);
   free(next);
   if(prev -> link == null)
   {
    last = prev;
   }
  }
 }
}
void display()
{
curr = first;
 while(curr != null)
 {
printf("\n %d", curr -> data);
curr = curr -> link;
 }
}
```

**Output:**

```
Singly linked list
1.create
2.insert
3.del
4.exit
```

```
Enter your choice 1
Enter the data:2
1.create
2.insert
3.del
4.exit

Enter your choice 2
Enter the data: 4
Enter the position: 2
2
4
1.create
2.insert
3.del
4.exit

Enter your choice 4
```

## Week: 2

**Write a C program to perform various operations such as creation, insertion, deletion and display on doubly linked list**

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

typedef struct dubll
{
 int data;
 struct dubll *leftlink,*rightlink;
}*DUBLL;

DUBLL high,temp_node,low,last,pntr;
int flag=0;

DUBLL NodeAlloc();
DUBLL Search(int,int);

void CreateItem();
void AppendItem();
void PrintItem();
void DeleteItem();
DUBLL Search(int item,int flag);
DUBLL NodeAlloc();
void InsertItem();

main(void)
```

```
{
 int choice,Item;
 high=NULL;
 while(1)
 {
  //clrscr();
printf("\n \t\t\t***** M A I N  M E N U *****\n\n");
printf("\n 1: Create Linked List \n 2: Append a Node to the List \n 3: Traverse the List \n 4: Delete a Node from the List \n 5: Search a Node \n 6: Insert a Node to the List \n 7: Close \n\n\t\t Enter your Option [ ]\b\b");
scanf("%d",&choice);
switch(choice)
  {
    case 1:
CreateItem();
        puts("\nPress any key to go back to main menu.");
getch();
        break;
    case 2:
AppendItem();
        break;
    case 3:
PrintItem();
        puts("\nPress any key to go back to main menu.");
getch();
     break;
    case 4:
DeleteItem();
        break;
    case 5:
printf("Find an Item: ");
scanf("%d",&Item);
temp_node=Search(Item,0);
        if(temp_node)
        {
      puts("The item is available in the Linked List.");
        }
        else
    {
         puts("The item is not found in the Linked List.");
         }
getch();
      break;
    case 6:
InsertItem();
        break;
    case 7:
        exit(0);
    default:
```

```
            puts("Invalid choice.");
            puts("\nPress any key to go back to main menu.");
getch();
            break;
   }
  }
}

/* Function to Create the list*/
void CreateItem()
{
   if(high==NULL)
   {
printf("\n --Creating the list--");
temp_node=NodeAlloc();
printf("\n Enter starting data (as integer value) :");
scanf("%d",&temp_node->data);
    high=temp_node;
   }
   else{ printf("\n List already created @ %d with %d as data.",high,high->data);}
}

/* Function to Append items to the list*/
void AppendItem()
{
   low=high;
   if(high==NULL)
   {
CreateItem();
   }
   else
   {
temp_node=NodeAlloc();
printf("\n Enter Item (in integer) :");
scanf("%d",&temp_node->data);
temp_node->rightlink=NULL;

     while(low->rightlink!=NULL)
      low=low->rightlink;
      low->rightlink=temp_node;
temp_node->leftlink=low;
      last=low->rightlink;

    }
}

/* Function to Traverse the list both ways and print the data*/
void PrintItem()
{
```

```
  DUBLL temp_node;
  if(high==NULL)
  {
printf("\n List is not available. Please create a list first.");
getch();
CreateItem();
  }
temp_node=high;
  last=low->rightlink;
printf("\n--Printing The List In Forward direction--\n");

  while(temp_node!=NULL)            //In forward direction
    {
printf("\t %d",temp_node->data);
temp_node = temp_node->rightlink;
    }
printf("\n");
printf("\n--Printing The List In Backward direction--\n");
temp_node=high;
    if(temp_node->rightlink==NULL){printf("%d",temp_node->data);return; }
      while(last!=NULL)            //In backward direction
       {
printf("\t %d",last->data);
         last = last->leftlink;
       }
}

/* Function to Delete items of the list*/
void DeleteItem()
{
 int value;
 DUBLL temp_node;
 if(high==NULL)
 {
printf("\n List is not available. Please create a list first.");
getch();
CreateItem();
 }
printf("\n Item to delete :");
scanf("%d",&value);
pntr=Search(value,1);
pntr->leftlink->rightlink=pntr->rightlink;
pntr->rightlink->leftlink=pntr->leftlink;
temp_node=pntr;
 free(temp_node);
}

/* Function to Search an item from the list*/
DUBLL Search(int item,int flag)
```

```
{
temp_node = high;
   if(high==NULL)
   {
printf("\n List is not available. Please create a list first.");
getch();
CreateItem();
   }
   while(temp_node!=NULL)
   {
    if(temp_node->data==item )
    {
     if(flag==0)
     {
       return(1);
     }
     else
     {
      return(temp_node);
     }
    }
temp_node=temp_node->rightlink;
   }
}

/* Function to Allocate nodes*/
DUBLL NodeAlloc()
{
  DUBLL tmep_node;
tmep_node=malloc(sizeof(struct dubll));
   if(tmep_node==NULL)
     {
printf("\n No memory available. Node allocation cannot be done.");
     }
tmep_node->rightlink=tmep_node->leftlink=NULL;
  return(tmep_node);
 }

/* Function to Insert items in the middle of the list*/
void InsertItem()
 {
  int node;
  DUBLL temp_node;

  if(high==NULL)
  {
printf("\n List is not available. Please create a list first.");
getch();
CreateItem();
```

```
  }
temp_node=NodeAlloc();
printf("Position At which node to be inserted:____& New Item Value:____");
scanf("%d",&node);
scanf("%d",&temp_node->data);
pntr=Search(node,1);

  if(pntr->rightlink==NULL){printf("\n The operation is not possible."); getch();return;}
temp_node->leftlink=pntr;                //creating link to new node
temp_node->rightlink=pntr->rightlink;

pntr->rightlink->leftlink=temp_node;
pntr->rightlink=temp_node;

printf("\n Item has been Inserted.");
getch();
 }
```

**Output:**

```
***** M A I N M E N U *****

 1: Create Linked List
 2: Append a Node to the List
 3: Traverse the List
 4: Delete a Node from the List
 5: Search a Node
 6: Insert a Node to the List
 7: Close

              Enter your Option [ 1]

 --Creating the list--
 Enter starting data (as integer value) :10

Press any key to go back to main menu.

                  ***** M A I N  M E N U *****


 1: Create Linked List
 2: Append a Node to the List
 3: Traverse the List
 4: Delete a Node from the List
 5: Search a Node
 6: Insert a Node to the List
 7: Close

              Enter your Option [ 2]
```

```
Enter Item (in integer) :20

                ***** M A I N M E N U *****


1: Create Linked List
2: Append a Node to the List
3: Traverse the List
4: Delete a Node from the List
5: Search a Node
6: Insert a Node to the List
7: Close

            Enter your Option [ 2]

Enter Item (in integer) :50

                ***** M A I N M E N U *****


1: Create Linked List
2: Append a Node to the List
3: Traverse the List
4: Delete a Node from the List
5: Search a Node
6: Insert a Node to the List
7: Close

            Enter your Option [ 3]

--Printing The List In Forward direction--
       10      20      50

--Printing The List In Backward direction--
       50      20      10
Press any key to go back to main menu.

                ***** M A I N M E N U *****


1: Create Linked List
2: Append a Node to the List
3: Traverse the List
4: Delete a Node from the List
5: Search a Node
6: Insert a Node to the List
7: Close

            Enter your Option [ 7 ]
```

# WEEK: 3

**Write a program that uses functions to perform the following operations on circular linked list.i) Creation ii) Insertion iii) Deletion iv) Traversal**

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

struct node
{
   int data;
   struct node *link;
};

struct node *head = NULL, *x, *y, *z;
void create();
void ins_at_beg();
void ins_at_pos();
void del_at_beg();
void del_at_pos();
void traverse();
void search();
void sort();
void update();
void rev_traverse(struct node *p);

void main()
{
   int ch;
printf("\n 1.Creation \n 2.Insertion at beginning \n 3.Insertion at remaining");
printf("\n4.Deletion at beginning \n5.Deletion at remaining \n6.traverse");
printf("\n7.Search\n8.sort\n9.update\n10.Exit\n");
   while (1)
   {
printf("\n Enter your choice:");
scanf("%d", &ch);
      switch(ch)
      {
      case 1:
         create();
         break;
      case 2:
```

```
ins_at_beg();
            break;
        case 3:
ins_at_pos();
            break;
        case 4:
del_at_beg();
            break;
        case 5:
del_at_pos();
            break;
        case 6:
            traverse();
            break;
        case 7:
            search();
            break;
        case 8:
            sort();
            break;
        case 9:
            update();
            break;
        case 10:
rev_traverse(head);
            break;
        default:
            exit(0);
        }
    }
}

void create()
{
    int c;

    x = (struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d", &x->data);
    x->link = x;
    head = x;
printf("\n If you wish to continue press 1 otherwise 0:");
scanf("%d", &c);
    while (c != 0)
    {
        y = (struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d", &y->data);
        x->link = y;
```

```
        y->link = head;
        x = y;
printf("\n If you wish to continue press 1 otherwise 0:");
scanf("%d", &c);
    }
}

void ins_at_beg()
{
    x = head;
    y = (struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d", &y->data);
    while (x->link != head)
    {
        x = x->link;
    }
    x->link = y;
    y->link = head;
    head = y;
}

void ins_at_pos()
{
    struct node *ptr;
    int c = 1, pos, count = 1;
    y = (struct node*)malloc(sizeof(struct node));
    if (head == NULL)
    {
printf("cannot enter an element at this place");
    }
printf("\n Enter the data:");
scanf("%d", &y->data);
printf("\n Enter the position to be inserted:");
scanf("%d", &pos);
    x = head;
ptr = head;
    while (ptr->link != head)
    {
        count++;
ptr = ptr->link;
    }
    count++;
    if (pos > count)
    {
printf("OUT OF BOUND");
        return;
    }
    while (c < pos)
```

```
    {
       z = x;
       x = x->link;
c++;
    }
    y->link = x;
    z->link = y;
}

void del_at_beg()
{
    if (head == NULL)
printf("\n List is empty");
    else
    {
       x = head;
       y = head;
       while (x->link !=  head)
       {
          x = x->link;
       }
       head = y->link;
       x->link = head;
       free(y);
    }
}
void del_at_pos()
{
    if (head == NULL)
printf("\n List is empty");
    else
    {
       int c = 1, pos;
printf("\n Enter the position to be deleted:");
scanf("%d", &pos);
       x = head;
       while (c < pos)
       {
          y = x;
          x = x->link;
c++;
       }
       y->link = x->link;
       free(x);
    }
}

void traverse()
{
```

```
   if (head == NULL)
printf("\n List is empty");
   else
   {
      x = head;
      while (x->link !=  head)
      {
printf("%d->", x->data);
         x = x->link;
      }
printf("%d", x->data);
   }
}

void search()
{
   int search_val, count = 0, flag = 0;
printf("\nenter the element to search\n");
scanf("%d", &search_val);
   if (head == NULL)
printf("\nList is empty nothing to search");
   else
   {
      x = head;
      while (x->link !=  head)
      {
         if (x->data == search_val)
         {
printf("\nthe element is found at %d", count);
            flag = 1;
            break;
         }
         count++;
         x = x->link;
      }
      if (x->data == search_val)
      {
printf("element found at postion %d", count);
      }
      if (flag == 0)
      {
printf("\nelement not found");
      }

   }
}

void sort()
{
```

```
    struct node *ptr, *nxt;
    int temp;
    if (head == NULL)
    {
printf("empty linkedlist");
    }
    else
    {
ptr = head;
        while (ptr->link !=  head)
        {
nxt = ptr->link;
           while (nxt !=  head)
           {
              if (nxt !=  head)
              {
                 if (ptr->data >nxt->data)
                 {
                    temp = ptr->data;
ptr->data = nxt->data;
nxt->data = temp;
                 }
              }
              else
              {
                 break;
              }
nxt = nxt->link;
           }
ptr = ptr->link;
        }
    }
}

void update()
{
    struct node *ptr;
    int search_val;
    int replace_val;
    int flag = 0;
    if (head == NULL)
    {
printf("\n empty list");
    }
    else
    {
printf("enter the value to be edited\n");
scanf("%d", &search_val);
fflush(stdin);
```

```
printf("enter the value to be replace\n");
scanf("%d", &replace_val);
ptr = head;
        while (ptr->link !=  head)
        {
           if (ptr->data == search_val)
           {
ptr->data = replace_val;
              flag = 1;
              break;
           }
ptr = ptr->link;
        }
        if (ptr->data == search_val)
        {
ptr->data = replace_val;
           flag = 1;
        }
        if (flag == 1)
        {
printf("\nUPdatesucessful");
        }
        else
        {
printf("\n update not successful");
        }
    }
}

void rev_traverse(struct node *p)
{
    int i = 0;

    if (head == NULL)
    {
printf("empty linked list");
    }
    else
    {
        if (p->link !=  head)
        {
i = p->data;
rev_traverse(p->link);
printf(" %d", i);
        }
        if (p->link == head)
        {
printf(" %d", p->data);
        }
```

```
    }
}
```

**output:**

```
1.Creation
2.Insertion at beginning
3.Insertion at remaining
4.Deletion at beginning
5.Deletion at remaining
6.traverse
7.Search
8.sort
9.update
10.Exit

Enter your choice:6
List is empty

Enter your choice:5
List is empty

Enter your choice:9
empty list

Enter your choice:7
enter the element to search
12
List is empty nothing to search

Enter your choice:1
Enter the data:10
If you wish to continue press 1 otherwise 0:0

Enter your choice:3
Enter the data:20
Enter the position to be inserted:5
OUT OF BOUND

Enter your choice:2
Enter the data:12

Enter your choice:6
12->10

Enter your choice:3
Enter the data:13
Enter the position to be inserted:3

Enter your choice:3
Enter the data:14
```

```
Enter the position to be inserted:4
Enter your choice:6
12->10->13->14

Enter your choice:3
Enter the data:24
Enter the position to be inserted:4

Enter your choice:6
12->10->13->24->14

Enter your choice:3
Enter the data:10
Enter the position to be inserted:100
OUT OF BOUND

Enter your choice:4

Enter your choice:6
10->13->24->14

Enter your choice:5
Enter the position to be deleted:4

Enter your choice:6
10->13->24

Enter your choice:5
Enter the position to be deleted:2

Enter your choice:6
10->24

Enter your choice:9
enter the value to be edited
23
enter the value to be replace
24
update not successful

Enter your choice:9
enter the value to be edited
24
enter the value to be replace
26
UPdatesucessful

Enter your choice:11
```

# Week:4

**a. Write C program to implement a stack using array**

```
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
   //clrscr();
   top=-1;
printf("\n Enter the size of STACK[MAX=100]:");
scanf("%d",&n);
printf("\n\t STACK OPERATIONS USING ARRAY");
printf("\n\t----------");
printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
   do
   {
printf("\n Enter the Choice:");
scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
         case 2:
         {
            pop();
            break;
         }
         case 3:
         {
            display();
            break;
         }
         case 4:
         {
printf("\n\t EXIT POINT ");
            break;
         }
```

```
            default:
            {
printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
            }

        }
    }
    while(choice!=4);
    return 0;
}
void push()
{
    if(top>=n-1)
    {
printf("\n\tSTACK is over flow");

    }
    else
    {
printf(" Enter a value to be pushed:");
scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
printf("\n\t Stack is under flow");
    }
    else
    {
printf("\n\t The popped elements is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
printf("\n%d",stack[i]);
printf("\n Press Next Choice");
    }
    else
    {
printf("\n The STACK is empty");
```

```
    }
}
```

**OUTPUT:**

```
Enter the size of STACK[MAX=100]:10

        STACK OPERATIONS USING ARRAY
       ----------------------------------
        1.PUSH
        2.POP
        3.DISPLAY
        4.EXIT
 Enter the Choice:1
 Enter a value to be pushed:12

 Enter the Choice:1
 Enter a value to be pushed:24

 Enter the Choice:1
 Enter a value to be pushed:98

 Enter the Choice:3

 The elements in STACK

98
24
12
 Press Next Choice
 Enter the Choice:2

        The popped elements is 98
 Enter the Choice:3

 The elements in STACK

24
12
 Press Next Choice
 Enter the Choice:4
        EXIT POINT
```

**b. Write a program that implement stack and its operations using Pointers**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX_SIZE 3
```

```
void push(int i);
void pop(void);

int choice, i;
int *tos, *p1, arr_stack[MAX_SIZE];
int exit_p = 1;

int main()
{
   int value;
tos = arr_stack; /* tos points to the top of stack */
   p1 = arr_stack; /* initialize p1 */
printf("\n Simple Stack Example - Pointers");
   do {
printf("\nStack Pointer : Main Menu");

printf("\n1.Push \t2.Pop \tOthers to exit : Your Choice : ");
scanf("%d", &choice);
       switch (choice) {
          case 1:
printf("Enter value: ");
scanf("%d", &value);
             push(value);
             break;
          case 2:
             pop();
             break;
          default:
exit_p = 0;
             break;
       }
   } while (exit_p);
   return 0;
}

void push(int i) {
   if (p1 == (tos + MAX_SIZE)) {
printf("\nStatus : Stack Overflow.\n");
   } else {
      *p1 = i;
printf("\nPush Value : %d ", *(p1));
      p1++;
   }
}

void pop(void) {
   if (p1 == tos) {
printf("\nStatus : Stack Underflow.\n");
```

```
        //return 0;
    } else {
        p1--;
printf("\nPop Value : %d ", *(p1));
    }
}
```

**Output:**

```
Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 1
Enter value: 100
Push Value : 100

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 1
Enter value: 200
Push Value : 200

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 1
Enter value: 300
Push Value : 300

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 1
Enter value: 400
Status : Stack Overflow.

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 1
Enter value: 500
Status : Stack Overflow.

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 1
Enter value: 2
Status : Stack Overflow.

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 2
Pop Value : 300

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 2
Pop Value : 200

Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 2
Pop Value : 100
```

```
Stack Pointer : Main Menu
1.Push 2.Pop Others to exit : Your Choice : 2
Status : Stack Underflow.

Stack Pointer : Main Menu
1.Push  2.Pop Others to exit : Your Choice : 3

--------------------
(program exited with code: 0)
```

# WEEK – 5

**a. Write a program that implement Queue (its operations) using Arrays?**

```
#include <stdio.h>
#define MAX 50

void    insert();
void   delete();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
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");
   }
}
```

**Output:**

```
1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit

Enter your choice : 1
```

```
Inset the element in queue : 10

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit

Enter your choice : 1
Inset the element in queue : 15

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit
Enter your choice : 1
Inset the element in queue : 20

1.Insert element to queue
2.Delete element from queue
3.Display all elements of queue
4.Quit

Enter your choice : 1
Inset the element in queue : 30

Enter your choice : 4
```

**b. Write a program that implement Queue (its operations) using Pointers**

```
#define true 1
#define false 0
#include<stdio.h>
#include<conio.h>
#include<process.h>

struct q_point
{
int ele;
struct q_point* n;
};
struct q_point *f_ptr = NULL;

int e_que(void);
void add_ele(int);
int rem_ele(void);
void show_ele();

/*main function*/
```

```
void main()
{
int ele,choice,j;
while(1)
{
clrscr();
printf("\n\n****IMPLEMENTATION OF QUEUE USING POINTERS****\n");
printf("==============================================");
printf("\n\t\t MENU\n");
printf("==============================================");
printf("\n\t[1] To insert an element");
printf("\n\t[2] To remove an element");
printf("\n\t[3] To display all the elements");
printf("\n\t[4] Exit");
printf("\n\n\tEnter your choice:");
scanf("%d", &choice);

switch(choice)
{
case 1:
{
printf("\n\tElement to be inserted:");
scanf("%d",&ele);
add_ele(ele);
getch();
break;
}
case 2:
{
if(!e_que())
{
j=rem_ele();
printf("\n\t%d is removed from the queue",j);
getch();
}
else
{
printf("\n\tQueue is Empty.");
getch();
}
break;
}
case 3:
show_ele();
getch();
break;
case  4:
exit(1);
break;
```

```
default:
printf("\n\tInvalid choice.");
getch();
break;
}
}
}

/* Function to check if the queue is empty*/
int e_que(void)
{
if(f_ptr==NULL)
return  true;
return false;
}

/* Function to add an element to the queue*/
void add_ele(int ele)
{
struct q_point *queue = (struct q_point*)malloc(sizeof(struct q_point));
queue->ele = ele;
queue->n = NULL;
if(f_ptr==NULL)
f_ptr = queue;
else
{
struct q_point* ptr;
ptr = f_ptr;
for(ptr=f_ptr ;ptr->n!=NULL; ptr=ptr->n);
ptr->n = queue;
}
}

/* Function to remove an element from the queue*/
int rem_ele()
{
struct q_point* queue=NULL;
if(e_que()==false)
{
int j = f_ptr->ele;
queue=f_ptr;
f_ptr = f_ptr->n;
free (queue);
return j;
}
else
{
printf("\n\tQueue is empty.");
return -9999;
```

```
}
}

/* Function to display the queue*/
void show_ele()
{
struct q_point *ptr=NULL;
ptr=f_ptr;
if(e_que())
{
printf("\n\tQUEUE is Empty.");
return;
}
else
{
printf("\n\tElements present in Queue are:\n\t");
while(ptr!=NULL)
{
printf("%d\t",ptr->ele);
ptr=ptr->n;
}
}
}
```

**Output:**

```
****IMPLEMENTATION OF QUEUE USING POINTERS****
==============================================
MENU
==============================================
[1] To insert an element
[2] To remove an element
[3] To display all the elements
[4] Exit

Enter your choice:1

Element to be inserted:23


****IMPLEMENTATION OF QUEUE USING POINTERS****
==============================================
MENU
==============================================
[1] To insert an element
[2] To remove an element
[3] To display all the elements
[4] Exit

Enter your choice:
3

Elements present in Queue are:
23


****IMPLEMENTATION OF QUEUE USING POINTERS****
==============================================
MENU
==============================================
[1] To insert an element
[2] To remove an element
[3] To display all the elements
[4] Exit
Enter your choice:2
23 is removed from the queue
****IMPLEMENTATION OF QUEUE USING POINTERS****
==============================================
MENU
==============================================
[1] To insert an element
[2] To remove an element
[3] To display all the elements
[4] Exit
```

```
Enter your choice:4
Exit
```

## Week: 6

Write a program that implements the following sorting methods to sort a given list of integers in ascending order

i) Bubble sort ii) Selection sort iii) Insertion sort

**C Program to Sort N Numbers in Ascending Order using Bubble Sort**

```
#include <stdio.h>
#include<conio.h>
#define MAXSIZE 10

void main()
{
   int array[MAXSIZE];
   int i, j, num, temp;

printf("Enter the value of num \n");
scanf("%d", &num);
printf("Enter the elements one by one \n");
   for (i = 0; i<num; i++)
   {
scanf("%d", &array[i]);
   }
printf("Input array is \n");
   for (i = 0; i<num; i++)
   {
printf("%d\n", array[i]);
   }
   /* Bubble sorting begins */
   for (i = 0; i<num; i++)
   {
      for (j = 0; j < (num - i - 1); j++)
      {
         if (array[j] > array[j + 1])
         {
            temp = array[j];
            array[j] = array[j + 1];
            array[j + 1] = temp;
         }
      }
   }
printf("Sorted array is...\n");
   for (i = 0; i<num; i++)
   {
printf("%d\n", array[i]);
   }
}
```

**Output**

```
Enter the value of num
          6
Enter the elements one by one
          23   45    67    89    12    34
```

```
Input array is
          23  45   67   89   12   34
Sorted array is...
12  23   34   45   67   89
```

**Selection sort**

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

void main()
{
int a[100],n,i,j,min,temp;
clrscr();
printf("\n Enter the Number of Elements: ");
scanf("%d",&n);
printf("\n Enter %d Elements: ",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
for(i=0;i<n-1;i++)
{
min=i;
for(j=i+1;j<n;j++)
{
if(a[min]>a[j])
min=j;
}
if(min!=i)
{
temp=a[i];
a[i]=a[min];
a[min]=temp;
}
}
printf("\n The Sorted array in ascending order: ");
for(i=0;i<n;i++)
{
printf("%d ",a[i]);
}
getch();
}
```

**Output:**

```
Enter the number of Elements:
        5
Enter 5 Elements:
        4 1  9  3  6
The stored Array in Ascending Order:
        1     3 4 6 9
```

**Insertion Sort**

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int arr[50], size, i, j, k, element, index;
printf("Enter Array Size: ");
scanf("%d", &size);
printf("Enter %d Array Elements: ", size);
   for(i=0; i<size; i++)
scanf("%d", &arr[i]);
   for(i=1; i<size; i++)
   {
      element = arr[i];
      if(element<arr[i-1])
      {
         for(j=0; j<=i; j++)
         {
            if(element<arr[j])
            {
               index = j;
               for(k=i; k>j; k--)
arr[k] = arr[k-1];
               break;
            }
         }
      }
      else
         continue;
arr[index] = element;
   }
printf("\nSorted Array:\n");
   for(i=0; i<size; i++)
printf("%d ", arr[i]);
getch();
   return 0;
```

```
}
```

**Output:**

```
Enter array size:
        5
Enter 5 array elements:
        28  16   5   11   0
Sorted Array:
        0  5   11   16   28
```

# WEEK - 7

Write a program that use both recursive and non-recursive functions to perform the following searching operations for a Key value in a given list of integers:

i) Linear search

ii) Binary search

**Linear Search**

```
#include <stdio.h>
#include <conio.h>
#define MAX_LEN 10
void l_search_recursive(int l[],int num,intele);
void l_search_nonrecursive(int l[],int num,intele);
void read_list(int l[],int num);
void print_list(int l[],int num);

int main()
{
   int l[MAX_LEN], num, ele;
   int ch;
   printf("=====================================================");
   printf("\n\t\t\tMENU");
   printf("\n=====================================================");
   printf("\n[1] Linear Search using Recursion method");
   printf("\n[2] Linear Search using Non-Recursion method");
   printf("\n\n Enter your Choice:");
   scanf("%d",&ch);
```

```
    if(ch<=2 &ch>0)
    {
       printf("Enter the number of elements :");
       scanf("%d",&num);
       read_list(l,num);
       printf("\nElements present in the list are:\n\n");
       print_list(l,num);
       printf("\n\nElement you want to search:\n\n");
       scanf("%d",&ele);

       switch(ch)
       {
       case 1:
          printf("\n**Recursion method**\n");
          l_search_recursive(l,num,ele);
          getch();
          break;

       case 2:
          printf("\n**Non-Recursion method**\n");
          l_search_nonrecursive(l,num,ele);
          getch();
          break;
       }
    }
    getch();
}

void l_search_nonrecursive(int l[],int num,intele)
{
    int j, f=0;
    for(j=0; j<num; j++)
       if( l[j] == ele)
       {
          printf("\nThe element %d is present at position %d in list\n",ele,j);
          f=1;
          break;
       }
    if(f==0)
       printf("\nThe element is %d is not present in the list\n",ele);
}

void l_search_recursive(int l[],int num,intele)
{
    int f = 0;

    if( l[num] == ele)
    {
       printf("\nThe element %d is present at position %d in list\n",ele,num);
```

```
        f=1;
    }
    else
    {
        if((num==0) && (f==0))
        {
            printf("The element %d is not found.",ele);
        }
        else
        {
            l_search_nonrecursive(l,num-1,ele);
        }
    }
    getch();
}

void read_list(int l[],int num)
{
    int j;
    printf("\nEnter the elements:\n");
    for(j=0; j<num; j++)
        scanf("%d",&l[j]);
}

void print_list(int l[],int num)
{
    int j;
    for(j=0; j<num; j++)
        printf("%d\t",l[j]);
}
```

**Binary search**

```
#include <stdio.h>
#define MAX_LEN 10
 void b_search_nonrecursive(int l[],int num,intele)
{
   int l1,i,j, flag = 0;
   l1 = 0;
   i = num-1;
   while(l1 <= i)
   {
      j = (l1+i)/2;
      if( l[j] == ele)
      {
    printf("\nThe element %d is present at position %d in list\n",ele,j);
         flag =1;
         break;
      }
```

```
      else
        if(l[j] <ele)
          l1 = j+1;
        else
          i = j-1;
    }
    if( flag == 0)
    printf("\nThe element %d is not present in the list\n",ele);
}

/* Recursive function*/
int b_search_recursive(int l[],int arrayStart,intarrayEnd,int a)
{
  int m,pos;
  if (arrayStart<=arrayEnd)
  {
    m=(arrayStart+arrayEnd)/2;
    if (l[m]==a)
      return m;
    else if (a<l[m])
      return b_search_recursive(l,arrayStart,m-1,a);
    else
      return b_search_recursive(l,m+1,arrayEnd,a);
    }
    return -1;
}

void read_list(int l[],int n)
{
    int i;
    printf("\nEnter the elements:\n");
    for(i=0;i<n;i++)
        scanf("%d",&l[i]);
}

void print_list(int l[],int n)
{
     int i;
    for(i=0;i<n;i++)
        printf("%d\t",l[i]);
}

/*main function*/
main()
{
    int l[MAX_LEN], num, ele,f,l1,a;
    int ch,pos;

    //clrscr();
```

```
  printf("=======================================================");
  printf("\n\t\t\tMENU");
  printf("\n=======================================================");
  printf("\n[1] Binary Search using Recursion method");
  printf("\n[2] Binary Search using Non-Recursion method");
  printf("\n\nEnter your Choice:");
  scanf("%d",&ch);

  if(ch<=2 &ch>0)
  {
   printf("\nEnter the number of elements : ");
   scanf("%d",&num);
   read_list(l,num);
   printf("\nElements present in the list are:\n\n");
   print_list(l,num);
   printf("\n\nEnter the element you want to search:\n\n");
   scanf("%d",&ele);


  switch(ch)
  {
   case 1:printf("\nRecursive method:\n");
      pos=b_search_recursive(l,0,num,ele);
      if(pos==-1)
      {
      printf("Element is not found");
      }
      else
      {
      printf("Element is found at %d position",pos);
      }
      //getch();
      break;
    case 2:printf("\nNon-Recursive method:\n");
      b_search_nonrecursive(l,num,ele);
      //getch();
      break;
   }
  }
//getch();
}
```

**OUTPUT:**

```
=====================================================
                    MENU
=====================================================
[1] Binary Search using Recursion method
[2] Binary Search using Non-Recursion method
Enter your Choice:1
Enter the number of elements : 5
Enter the elements:
12
22
32
42
52
Elements present in the list are:
12      22      32      42      52
Enter the element you want to search:
42
Recursive method:
Element is found at 3 position
```

**Write a C program to implement the tree traversal methods.**

```
#include  <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node* left;
    struct node* right;
};
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

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

```
{
    if (node == NULL)
        return;
printInorder(node->left);
printf("%d ", node->data);
printInorder(node->right);
}
void printPreorder(struct node* node)
{
    if (node == NULL)
        return;
printf("%d  ",  node->data);
printPreorder(node->left);
printPreorder(node->right);
}
  int main()
{
    struct node *root = newNode(1);
    root->left          = newNode(2);
    root->right         =  newNode(3);
    root->left->left   =   newNode(4);
    root->left->right  = newNode(5);

printf("\nPreorder traversal of binary tree is \n");
printPreorder(root);

printf("\nInorder traversal of binary tree is \n");
printInorder(root);

printf("\nPostorder traversal of binary tree is \n");
printPostorder(root);

getchar();
    return 0;
}
```

**Output:**

```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1Post-order traversal: 10 19 14 31 42 35 27
```

## WEEK – 8

# WEEK – 9

**Write a program to implement the graph traversal methods.**

```
#include<stdio.h>
void dfs();
void bfs();

struct link
{
struct node *next;
struct link *adj;
};

struct node
{
int data,status;
struct node *next;
struct link *adj;
};

struct node *start,*p,*q;
struct link *l,*k;

void create()
{
int dat,flag=0;dat=1;
while(dat)
{
scanf("%d",&dat);
 if(dat==0)
break;
p=(struct node*)malloc(sizeof(struct node
p->status=0;
p->next=NULL;
 if(flag==0)
{
start=p;
q=p;
flag++;
}
{
q->next=p;
q=p;
}
}
while(p!=NULL)
```

```
{
printf("Enter the links to %d (0 to end) : ",p->data);
flag=0;
while(1)
{
scanf("%d",&dat);
if(dat==0)
break;
k=(struct link*)malloc(sizeof(struct link));
if(flag==0)
{
p->adj=k;
l=k;
flag++;
}
else
{
l->adj=k;
l=k;
}
q=start;
while(q!=NULL)
{
if(q->data==dat)
k->next=q;
q=q->next;
}
}
p=p->next;
}
}

void bfs()
{
int q[20],i=1,j=0;
p=start;
while(p!=NULL)
{
p->status=0;
p=p->next;
}
p=start;
q[0]=p->data;
p->status=1;
while(1)
{
if(q[j]==0)
break;
p=start;
```

```
while(p!=NULL)
{
if(p->data==q[j])
break;
p=p->next;
}
k=p->adj;
while(k!=NULL)
{
q=k->next;
if(q->status==0)
{
q[i]=q->data;
q->status=1;
q[i+1]=0;
i++;
}
k=k->adj;
}
j++;
}
j=0;
printf("Breadth First Search Results\n");
while(q[j]!=0)
{
printf("%d ",q[j]);
j++;
}
getche();

}
void dfs()
{
int stack[20],top=1;
printf("Deapth First Search Results");
p=start;
while(p!=NULL)
{
p->status=0;
p=p->next;
}
p=start;
stack[0]=0;
stack[top]=p->data;
p->status=1;
while(1)
{
if(stack[top]==0)
break;
```

```
p=start;
while(p!=NULL)
{
if(p->data==stack[top])
break;
p=p->next;
}
printf("%d ",stack[top]);
 top--;
k=p->adj;
while(k!=NULL)
{
q=k->next;
if(q->status==0)
{
top++;
stack[top]=q->data;
q->status=1;
}
k=k->adj;
}
}
}

int main()
{
int ch;
create();
while(1)
{
printf("1: DFS\n2: BSF\n0: Exit\nEnter your choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1:
dfs();
break;
case 2:
bfs();
break;
case 0:
exit(0);
break;
default:
printf("Incorrect choice!");
}
}
return 0;
}
```

# VIVA QUESTIONS

**1) What is data structure?**

Data structure refers to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with the data structure, we not only focus on one piece of data but the different set of data and how they can relate to one another in an organized manner.

**2) Differentiate between file and structure storage structure.**

The key difference between both the data structure is the memory area that is being accessed. When dealing with the structure that resides the main memory of the computer system, this is referred to as storage structure. When dealing with an auxiliary structure, we refer to it as file structures.

**3) When is a binary search best applied?**

A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is searched starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

**4) What is a linked list?**

A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link for data storage.

**5) How do you reference all the elements in a one-dimension array?**

To reference all the elements in an one -dimension array, you need to use an indexed loop, So that, the counter runs from 0 to the array size minus one. In this manner, You can reference all the elements in sequence by using the loop counter as the array subscript.

**6) In what areas do data structures are applied?**

Data structures are essential in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating

system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

**7) What is LIFO?**

LIFO is a short form of Last In First Out. It refers how data is accessed, stored and retrieved. Using this scheme, data that was stored last should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

**8 ) What is a queue?**

A queue is a data structure that can simulate a list or stream of data. In this structure, new elements are inserted at one end, and existing elements are removed from the other end.

**9) What are binary trees?**

A binary tree is one type of data structure that has two nodes, a left node, and a right node. In programming, binary trees are an extension of the linked list structures.

**10) Which data structures are applied when dealing with a recursive function?**

Recursion, is a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

**11) What is a stack?**

A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

**12) Explain Binary Search Tree**

A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

**13) What are multidimensional arrays?**

Multidimensional arrays make use of multiple indexes to store data. It is useful when storing data that cannot be represented using single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.

**14) Are linked lists considered linear or non-linear data structures?**

It depends on where you intend to apply linked lists. If you based it on storage, a linked list is considered non-linear. On the other hand, if you based it on access strategies, then a linked list is considered linear.

**15) How does dynamic memory allocation help in managing data?**

Apart from being able to store simple structured data types, dynamic memory allocation can combine separately allocated structured blocks to form composite structures that expand and contract as needed.

**16) What is FIFO?**

FIFO stands for First-in, First-out, and is used to represent how data is accessed in a queue. Data has been inserted into the queue list the longest is the one that is removed first.

**17) What is an ordered list?**

An ordered list is a list in which each node's position in the list is determined by the value of its key component, so that the key values form an increasing sequence, as the list is traversed.

**18) What is merge sort?**

Merge sort, is a divide-and-conquer approach for sorting the data. In a sequence of data, adjacent ones are merged and sorted to create bigger sorted lists. These sorted lists are then merged again to form an even bigger sorted list, which continues until you have one single sorted list.

**19) Differentiate NULL and VOID**

Null is a value, whereas Void is a data type identifier. A variable that is given a Null value indicates an empty value. The void is used to identify pointers as having no initial size.

**20) What is the primary advantage of a linked list?**

A linked list is an ideal data structure because it can be modified easily. This means that editing a linked list works regardless of how many elements are in the list.

**21) What is the difference between a PUSH and a POP?**

Pushing and popping applies to the way data is stored and retrieved in a stack. A push denotes data being added to it, meaning data is being "pushed" into the stack. On the other hand, a pop denotes data retrieval, and in particular, refers to the topmost data being accessed.

**22) What is a linear search?**

A linear search refers to the way a target key is being searched in a sequential data structure. In this method, each element in the list is checked and compared against the target key. The process is repeated until found or if the end of the file has been reached.

**23) How does variable declaration affect memory allocation?**

The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

**24) What is the advantage of the heap over a stack?**

The heap is more flexible than the stack. That's because memory space for the heap can be dynamically allocated and de-allocated as needed. However, the memory of the heap can at times be slower when compared to that stack.

**25) What is a postfix expression?**

A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

**26) What is Data abstraction?**

Data abstraction is a powerful tool for breaking down complex data problems into manageable chunks. This is applied by initially specifying the data objects involved and the operations to be

performed on these data objects without being overly concerned with how the data objects will be represented and stored in memory.

**27) How do you insert a new item in a binary search tree?**

Assuming that the data to be inserted is a unique value (that is, not an existing entry in the tree), check first if the tree is empty. If it's empty, just insert the new item in the root node. If it's not empty, refer to the new item's key. If it's smaller than the root's key, insert it into the root's left subtree, otherwise, insert it into the root's right subtree.

**28) How does a selection sort work for an array?**

The selection sort is a fairly intuitive sorting algorithm, though not necessarily efficient. In this process, the smallest element is first located and switched with the element at subscript zero, thereby placing the smallest element in the first position.

The smallest element remaining in the subarray is then located next to subscripts 1 through n-1 and switched with the element at subscript 1, thereby placing the second smallest element in the second position. The steps are repeated in the same manner till the last element.

**29) How do signed and unsigned numbers affect memory?**

In the case of signed numbers, the first bit is used to indicate whether positive or negative, which leaves you with one bit short. With unsigned numbers, you have all bits available for that number. The effect is best seen in the number range (an unsigned 8-bit number has a range 0-255, while the 8-bit signed number has a range -128 to +127.

**30) What is the minimum number of nodes that a binary tree can have?**

A binary tree can have a minimum of zero nodes, which occurs when the nodes have NULL values. Furthermore, a binary tree can also have 1 or 2 nodes.

**31) What are dynamic data structures?**

Dynamic data structures are structures that expand and contract as a program runs. It provides a flexible means of manipulating data because it can adjust according to the size of the data.

**32) In what data structures are pointers applied?**

Pointers that are used in linked list have various applications in the data structure. Data structures that make use of this concept include the Stack, Queue, Linked List and Binary Tree.

**33) Do all declaration statements result in a fixed reservation in memory?**

Most declarations do, with the exemption of pointers. Pointer declaration does not allocate memory for data, but for the address of the pointer variable. Actual memory allocation for the data comes during run-time.

**34) What are ARRAYs?**

When dealing with arrays, data is stored and retrieved using an index that refers to the element number in the data sequence. This means that data can be accessed in any order. In programming, an array is declared as a variable having a number of indexed elements.

**35) What is the minimum number of queues needed when implementing a priority queue?**

The minimum number of queues needed in this case is two. One queue is intended for sorting priorities while the other queue is used for actual storage of data.

**36) Which sorting algorithm is considered the fastest?**

There are many types of sorting algorithms: quick sort, bubble sort, balloon sort, radix sort, merge sort, etc. Not one can be considered the fastest because each algorithm is designed for a particular data structure and data set. It would depend on the data set that you would want to sort.

**37) Differentiate STACK from ARRAY.**

Stack follows a LIFO pattern. It means that data access follows a sequence wherein the last data to be stored when the first one to be extracted. Arrays, on the other hand, does not follow a particular order and instead can be accessed by referring to the indexed element within the array.

**38) Give a basic algorithm for searching a binary search tree.**

if the tree is empty, then the target is not in the tree, end search

2. if the tree is not empty, the target is in the tree
3. check if the target is in the root item
4. if a target is not in the root item, check if a target is smaller than the root's value
5. if a target is smaller than the root's value, search the left subtree
6. else, search the right subtree

**39) What is a dequeue?**

A dequeue is a double-ended queue. This is a structure wherein elements can be inserted or removed from either end.

**40) What is a bubble sort and how do you perform it?**

A bubble sort is one sorting technique that can be applied to data structures such as an array. It works by comparing adjacent elements and exchanges their values if they are out of order. This method lets the smaller values "bubble" to the top of the list, while the larger value sinks to the bottom.

**41) What are the parts of a linked list?**

A linked list typically has two parts: the head and the tail. Between the head and tail lie the actual nodes. All these nodes are linked sequentially.

**42) How does selection sort work?**

Selection sort works by picking the smallest number from the list and placing it at the front. This process is repeated for the second position towards the end of the list. It is the simplest sort algorithm.

**43) What is a graph?**

A graph is one type of data structure that contains a set of ordered pairs. These ordered pairs are also referred to as edges or arcs and are used to connect nodes where data can be stored and retrieved.

**44) Differentiate linear from a nonlinear data structure.**

The linear data structure is a structure wherein data elements are adjacent to each other. Examples of linear data structure include arrays, linked lists, stacks, and queues. On the other hand, a non-linear data structure is a structure wherein each data element can connect to more than two adjacent data elements. Examples of nonlinear data structure include trees and graphs.

**45) What is an AVL tree?**

An AVL tree is a type of binary search tree that is always in a state of partially balanced. The balance is measured as a difference between the heights of the subtrees from the root. This self-balancing tree was known to be the first data structure to be designed as such.

**46) What are doubly linked lists?**

Doubly linked lists are a special type of linked list wherein traversal across the data elements can be done in both directions. This is made possible by having two links in every node, one that links to the next node and another one that connects to the previous node.

**47) What is Huffman's algorithm?**

Huffman's algorithm is used for creating extended binary trees that have minimum weighted path lengths from the given weights. It makes use of a table that contains the frequency of occurrence for each data element.

**48) What is Fibonacci search?**

Fibonacci search is a search algorithm that applies to a sorted array. It makes use of a divide-and-conquer approach that can significantly reduce the time needed in order to reach the target element.

**49) Briefly explain recursive algorithm.**

Recursive algorithm targets a problem by dividing it into smaller, manageable sub-problems. The output of one recursion after processing one sub-problem becomes the input to the next recursive process.

**50) How do you search for a target key in a linked list?**

To find the target key in a linked list, you have to apply sequential search. Each node is traversed and compared with the target key, and if it is different, then it follows the link to the next node. This traversal continues until either the target key is found or if the last node is reached.