NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE

(NAAC Accredited)

(Approved by AICTE , Affiliated to APJ Abdul Kalam Technological University, Kerala)

Pampady, Thiruvilwamala(PO), Thrissur(DT), Kerala 680 588

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

CS231 DATA STRUCTURE LAB

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT THE DEPARTMENT

- Established in: 2002

- Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security

- Approved by AICTE New Delhi and Accredited by NAAC
- Certified by ISO 9001-2015
- Affiliated to A P J Abdul Kalam Technological University, Kerala.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Team work and leadership qualities.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex

engineering problems.

2. **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.
3. **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.
4. **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.
5. **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.
6. **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.
7. **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.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **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.
11. **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.
12. **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 (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high

quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOME

CO 1	Design and implement programs on 8086 microprocessor
CO 2	To provide solid foundation on interfacing the external devices to the processor according to the user requirements
CO 3	Design and implement 8051 microcontroller based systems
CO 4	To Understand the concepts related to I/O and memory interfacing
CO 5	To learn about interfacing stepper motor working and its interfacing
CO 6	To learn about generation of waveforms using microcontroller
CO 7	To learn about different types of flag registers and their changes while performing arithmetic operations

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO 1	1	**3**	**3**	**3**	**3**	**3**						
CO 2	**2**	**1**	**1**	**2**	**1**			**2**	**2**	**1**	**3**	**2**
CO 3	**1**	**1**	**2**	**2**	**3**		**3**	**3**	**3**	**3**	**3**	**3**
CO 4	**1**							**2**	**2**	**1**	**3**	**2**
CO 5						**3**						
CO 6												**3**
CO 7	1	**2**	**2**	**1**	**2**	**3**	**1**	**1**		**3**		**1**

MAPPING OF COURSE OUTCOMES WITH PROGRAM SPECIFIC OUTCOMES

	PSO1	PSO2	PSO3
CO1	3	3	-
CO2	3	3	-
CO3	3	3	-
CO4	3	3	-
CO5	3	3	-
CO6	3	3	-

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

PREPARATION FOR THE LABORATORY SESSION
GENERAL INSTRUCTIONS TO STUDENTS

1. Read carefully and understand the description of the experiment in the lab manual. You may go to the lab at an earlier date to look at the experimental facility and understand it better. Consult the appropriate references to be completely familiar with the concepts and hardware.

2. Make sure that your observation for previous week experiment is evaluated by the faculty member and you have transferred all the contents to your record before entering to the lab/workshop.

3. At the beginning of the class, if the faculty or the instructor finds that a student is not adequately prepared, they will be marked as absent and not be allowed to perform the experiment.

4. Bring necessary material needed (writing materials, graphs, calculators, etc.) to perform the required preliminary analysis. It is a good idea to do sample calculations and as much of the analysis as possible during the session. Faculty help will be available. Errors in the procedure may thus be easily detected and rectified.

5. Please actively participate in class and don't hesitate to ask questions. Please utilize the teaching assistants fully. To encourage you to be prepared and to read the lab manual before coming to the laboratory, unannounced questions may be asked at any time during the lab.

6. Carelessness in personal conduct or in handling equipment may result in serious injury to the individual or the equipment. Do not run near moving machinery/equipment. Always be on the alert for strange sounds. Guard against entangling clothes in moving parts of machinery.

7. Students must follow the proper dress code inside the laboratory. To protect clothing from dirt, wear a lab coat. Long hair should be tied back. Shoes covering the whole foot will have to be worn.

8. In performing the experiments, please proceed carefully to minimize any water spills, especially on the electric circuits and wire.

9. Maintain silence, order and discipline inside the lab. Don't use cell phones inside the laboratory.

10. Any injury no matter how small must be reported to the instructor immediately.

11. Check with faculty members one week before the experiment to make sure that you have the handout for that experiment and all the apparatus.

AFTER THE LABORATORY SESSION

1. Clean up your work area.

2. Check with the technician before you leave.

3. Make sure you understand what kind of report is to be prepared and due submission of record is next lab class.

4. Do sample calculations and some preliminary work to verify that the experiment was successful

MAKE-UPS AND LATE WORK

Students must participate in all laboratory exercises as scheduled. They must obtain permission from the faculty member for absence, which would be granted only under justifiable circumstances. In such an event, a student must make arrangements for a make-up laboratory, which will be scheduled when the time is available after completing one cycle. Late submission will be awarded less mark for record and internals and zero in worst cases.

LABORATORY POLICIES

1. Food, beverages & mobile phones are not allowed in the laboratory at any time.

2. Do not sit or place anything on instrument benches.

3. Organizing laboratory experiments requires the help of laboratory technicians and staff. Be punctual.

SYLLABUS

Course No.	Course Name	L-T-P-Credits	Year of Introduction
CS231	**DATA STRUCTURES LAB**		**2015**

Course Objective:

- To implement basic linear and non-linear data structures and their major operations.
- To implement applications using these data structures.
- To implement algorithms for various sorting techniques.

List of Exercises / Experiments
(For Computer Science and Engineering Branch)

List of Exercises/Experiments : (Minimum 12 are to be done)
1. Implementation of Stack and Multiple stacks using one dimensional array. **
2. Application problems using stacks: Infix to post fix conversion, postfix and pre-fix evaluation, MAZE problem etc. **
3. Implementation of Queue, DEQUEUE and Circular queue using arrays.
4. Implementation of various linked list operations. **
5. Implementation of stack, queue and their applications using linked list.
6. Implementation of trees using linked list
7. Representation of polynomials using linked list, addition and multiplication of polynomials. **
8. Implementation of binary trees using linked lists and arrays- creations, insertion, deletion and traversal. **
9. Implementation of binary search trees – creation, insertion, deletion, search
10. Application using trees
11. Implementation of sorting algorithms – bubble, insertion, selection, quick (recursive and non-recursive), merge sort (recursive and non-recursive), and heap sort.**
12. Implementation of searching algorithms – linear search, binary search.**
13. Representation of graphs and computing various parameters (in degree, out degree etc.) - adjacency list, adjacency matrix.
14. Implementation of BFS, DFS for each representation.
15.
16. Implementation of various string operations
17. Simulation of first-fit, best-fit and worst-fit allocations.
18. Simulation of a basic memory allocator and garbage collector using doubly linked list.
** mandatory

Course Outcome

- appreciate the importance of structure and abstract data type, and their basic usability in different applications
- analyze and differentiate different algorithms based on their time complexity.

- implement linear and non-linear data structures using linked lists.
- understand and apply various data structure such as stacks, queues, trees, graphs, etc. to solve various computing problems.
- implement various kinds of searching and sorting techniques, and decide when to choose which technique.
- 6. identify and use a suitable data structure and algorithm to solve a real world problem.

INDEX

EXPERIMENT – 1
Stack and Multiple stacks using one dimensional array

AIM

Implementation of Stack and Multiple stacks using one dimensional array

ALGORITHM

Step 1. Here we use 2 arrays min[] and max[] to represent the lower

and upper bounds for a stack

Step 2. Array s[] stores the elements of the stack

Step 3. Array top[] is used to store the top index for each stack

Step 4. Variable ns represents the stack number

Step 5. Variable size represents the size for each stack in an array

Step 6. First we build a function init() to initialize the starting values

Step 7. Then we have a function createstack() to create the stack

Step 8. Function Push() & Pop() are used to push and pop an element to and from the stack

Step 9. Function Display() is used to display the elements in a particular stack

0

PROGRAM

```
#include <stdio.h>

#include <stdlib.h>/* run this program using the console pauser or add your own getch, system("pause") or input loop */

//min represents the lower bound for a stack

//max represents the upper bound for a stackint s[50],top[50],min[50],max[50];//ns is stack number

//size is size of the stack

int ns,size;//function to initialize starting values of top,min,max and stack

void init(void)
```

```
{
 int i;
 for(i=0;i<50;++i)
 {
 s[i]=min[i]=max[i] = 0;
 top[i]=-1;
 }//end for}//end function//function to create the stack
void createstack()
{
 int i ;
 //min and top of 0th stack will be -1
 //and it's max will be at index one lesser than it's sizemin[0]= -1;
 max[0] = size -1;
 top[0]=-1;
  //min and top of 1,2,3,….th stacks
 for(i=1;i<ns;++i)
 {
 min[i]= min[i-1] + size;
 top[i] = min [i];}//end for
  //max of 1,2,3,….th stacks will me min of 2,3,4,….th stack
 for(i=1;i<ns;++i)
 {
 max[i]= min[i+1];
  }//end for
 }//end function//function to push element to stack
```

```
//parameters passed will be the item user wants to push and the stack no. to //push
void push(int ele,int k)
{
 //check for stack overflow
 if(top[k-1]==max[k-1])
 {
 printf("Stack no %d is full i.e overflow\n",k);
 return;
 }//end if
  ++top[k-1];
 s[top[k-1]] = ele;
}//end function push//function to pop an element form stack
//parameters passed is the stack no. from which we want to pop an element
void pop(int k)
{
 //check for underflow
 if(top[k-1]==min[k-1])
 {
 printf("\nStack no %d is empty i.e underflow\n",k);
 return;}//end if
  //else delete the item
 printf("%d from stack %d is deleted:\n",s[top[k-1]],k);
- -top[k-1];
 }//end function pop
  //function to display any stack
```

```
//parameter passed is the stack number to display
void display(int k)
{
 //first check for stack empty condition
 //variable j is used to iterate through the list
 int j;
 if(top[k-1]==min[k-1])
 {
 printf("\nStack no %d is empty\n",k);
 return;
 }//end if

 //else display the list
 printf("\nStack %d →> ",k);

 for(j=min[k-1]+1;j<=top[k-1];++j)
 {
 printf("%d",s[j] );
 }//end for

} //end function display//main functionint main() {
 //variable choice,stack number and item to push is initialized
 int ele,ch,skn;
 init();//function call
```

```
//input the number of stacks
printf("\nEnter the number of Stacks\n");
scanf("%d",&ns);

//size of each stack
//size = size of array/number of stacks
size = 50/ns;
 createstack();//function call
printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
do{
//ask for users choice
printf("\nEnter your choice : \t");
scanf("%d",&ch);
 switch(ch)
{
case 1: printf("\nEnter the stack no : \t");
scanf("%d",&skn);
printf("\nEnter the element : \t");
scanf("%d",&ele);
push(ele,skn);
break;
 case 2 : printf("\nEnter the stack no to pop : \t");
scanf("%d",&skn);
pop(skn);
break;
```

```
 case 3: printf("\nEnter the stack no to display : \t");
scanf("%d",&skn);
display(skn);
break;
 case 4 : printf("\nProgram Terminating");
break;
 default : printf("\nInvalid Option\n");
}//end switch
 }//end do-while loop
while(ch!=4);
 return 0;
}
```

OUTPUT

RESULT AND DISCUSSION

VIVA QUESTIONS

.

EXPERIMENT – 2

Application problems using stacks

AIM

Application problems using stacks: Infix to post fix conversion, postfix and pre-fix evaluation, MAZE problem etc.

ALGORITHM

Step 1. Scan the infix expression from left to right.

Step 2. If the scanned character is an operand, output it.

Step 3. Else,

1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.

2 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you

encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

Step 4. If the scanned character is an '(', push it to the stack.

Step 5. If the scanned character is an ')', pop the stack and output it until a '(' is encountered,and discard both the parenthesis.

Step 6. Repeat steps 2-6 until infix expression is scanned.

Step 7. Print the output

Step 8. Pop and output from the stack until it is not empty.

Step 9. Stop

PROGRAM

```
#include<bits/stdc++.h>

using namespace std;

 //Function to return precedence of operators
```

```
int prec(char c) {
    if(c == '^')
        return 3;
    else if(c == '*' || c == '/')
        return 2;
    else if(c == '+' || c == '-')
        return 1;
    else
        return -1;
}
  // The main function to convert infix expression
//to postfix expression
void infixToPostfix(string s) {
stack<char> st; //For stack operations, we are using C++ built in stack
    string result;
      for(int i = 0; i < s.length(); i++) {
        char c = s[i];
          // If the scanned character is
        // an operand, add it to output string.
        if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <= '9'))
            result += c;
          // If the scanned character is an
        // ‘(‘, push it to the stack.
        else if(c == '(')
            st.push('(');
```

```
        // If the scanned character is an ‘)’,
      // pop and to output string from the stack
      // until an ‘(‘ is encountered.
      else if(c == ')') {
         while(!st.empty() && st.top() != '(')
         {
            char temp = st.top();
            st.pop();
            result += temp;
         }
         st.pop();
      }

      //If an operator is scanned
      else {
         while(!st.empty() && prec(s[i]) < prec(st.top())) {
            char temp = st.top();
st.pop();
            result += temp;
         }
         st.push(c);
      }
   }
    // Pop all the remaining elements from the stack
   while(!st.empty()) {
```

```
        char temp = st.top();
        st.pop();
        result += temp;
    }
     cout << result << endl;
}
  //Driver program to test above functions
int main() {
    string exp = "a+b*(c^d-e)^(f+g*h)-i";
    infixToPostfix(exp);
    return 0;
}
```

OUTPUT

RESULT AND DISCUSSION

VIVA QUESTIONS

EXPERIMENT – 3

Various linked list operations

AIM

Implementation of various linked list operations

ALORITHM

Insertion

Step 1: Allocate the space for the new node PTR

Step 2: SET PTR -> DATA = VAL

Step 3: IF FRONT = NULL

SET FRONT = REAR = PTR

SET FRONT -> NEXT = REAR -> NEXT = NULL

ELSE

SET REAR -> NEXT = PTR

SET REAR = PTR

SET REAR -> NEXT = NULL

[END OF IF]

Step 4: END

Deletion

Step 1: IF FRONT = NULL

Write " Underflow "

Go to Step 5

[END OF IF]

Step 2: SET PTR = FRONT

Step 3: SET FRONT = FRONT -> NEXT

Step 4: FREE PTR

Step 5: END

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h> struct node
{
int info;
struct node *next;
};
typedef struct node NODE; NODE *start;
void createmptylist(NODE **start)
{
*start=(NODE *)NULL;
}
void traversinorder(NODE *start)
{
while(start != (NODE *) NULL)
{
printf("%d\n",start->info); start=start->next;
}
}
void insertatbegin(int item)
{
NODE *ptr;
```

```
ptr=(NODE *)malloc(sizeof(NODE)); ptr->info=item;
if(start==(NODE *)NULL) ptr->next=(NODE *)NULL; else
ptr->next=start; start=ptr;
}

void insert_at_end(int item)
{
NODE *ptr,*loc;
ptr=(NODE *)malloc(sizeof(NODE)); ptr->info=item;
ptr->next=(NODE *)NULL; if(start==(NODE*)NULL) start=ptr;
else
{
loc=start;
while(loc->next!=(NODE *)NULL) loc=loc->next;
loc->next=ptr;
}
}
void insert_spe(NODE *start,int item)
{
NODE *ptr,*loc; int temp,k;
for(k=0,loc=start;k<temp;k++)
{
loc=loc->next; if(loc==NULL)
{
printf("node in the list at less than one\n"); return;
```

```
}

}

ptr=(NODE *)malloc(sizeof(NODE)); ptr->info=item;

ptr->next=loc->next;; loc->next=ptr;

}

void main()

{

int choice,item,after; char ch;

clrscr(); createmptylist(start); do

{

printf("1.Insert element at begin \n"); printf("2. insert element at end positon\n");
printf("3. insert specific the position\n"); printf("4.travers the list in order\n"); printf("5.
exit\n");

printf("enter your choice\n"); scanf("%d",&choice); switch(choice)

{

case 1: printf("Enter the item\n"); scanf("%d",&item); insertatbegin(item);       break;

case 2: printf("Enter the item\n"); scanf("%d",&item);


insert_at_end(item); break;

case 3: printf("Enter the item\n"); scanf("%d",&item); insert_spe(start,item); break;

case 4: printf("\ntravers the list\n"); traversinorder(start);           break;

case 5: return;

}

fflush(stdin);

printf("do your want continous\n"); scanf("%c",&ch);

}while((ch='y')||(ch='y')); getch();
```

}

OUTPUT

RESULT AND DISCUSSION

VIVA QUESTIONS

EXPERIMENT – 4

Representation of polynomials using linked list, addition and multiplication of polynomials

AIM

Representation of polynomials using linked list, addition and multiplication of polynomials

ALORITHM

Step 1: Start

Step 2: Initialize first polynomial

Step 3: Initialize second polynomial

Step 4: perform Polynomial addition

Step 5: Perform Polynomial multiplication

Step 6: Display results

Step 7:Stop

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h> struct node
{
int info;
struct node *next;
};
typedef struct node NODE; NODE *start;
void createmptylist(NODE **start)
{
*start=(NODE *)NULL;
```

```
}
void traversinorder(NODE *start)
{
while(start != (NODE *) NULL)
{
printf("%d\n",start->info); start=start->next;
}
}
void insertatbegin(int item)
{
NODE *ptr;
ptr=(NODE *)malloc(sizeof(NODE)); ptr->info=item;
if(start==(NODE *)NULL) ptr->next=(NODE *)NULL; else
ptr->next=start; start=ptr;
}

void insert_at_end(int item)
{
NODE *ptr,*loc;
ptr=(NODE *)malloc(sizeof(NODE)); ptr->info=item;
ptr->next=(NODE *)NULL; if(start==(NODE*)NULL) start=ptr;
else
{
loc=start;
while(loc->next!=(NODE *)NULL) loc=loc->next;
```

```
loc->next=ptr;
}
}
void insert_spe(NODE *start,int item)
{
NODE *ptr,*loc; int temp,k;
for(k=0,loc=start;k<temp;k++)
{
loc=loc->next; if(loc==NULL)
{
printf("node in the list at less than one\n"); return;
}
}
ptr=(NODE *)malloc(sizeof(NODE)); ptr->info=item;
ptr->next=loc->next;; loc->next=ptr;
}
void main()
{
int choice,item,after; char ch;
clrscr(); createmptylist(start); do
{
printf("1.Insert element at begin \n"); printf("2. insert element at end positon\n");
printf("3. insert specific the position\n"); printf("4.travers the list in order\n"); printf("5.
exit\n");
printf("enter your choice\n"); scanf("%d",&choice); switch(choice)
{
```

```
case 1: printf("Enter the item\n"); scanf("%d",&item); insertatbegin(item);       break;
case 2: printf("Enter the item\n"); scanf("%d",&item);
 insert_at_end(item); break;
case 3: printf("Enter the item\n"); scanf("%d",&item); insert_spe(start,item); break;
case 4: printf("\ntravers the list\n"); traversinorder(start);          break;
case 5: return;
}
fflush(stdin);
printf("do your want continous\n"); scanf("%c",&ch);
}while((ch='y')||(ch='y')); getch();
}
```

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 5

Implementation of binary trees using linked lists and arrays- creations, insertion, deletion and traversal

AIM

Implementation of binary trees using linked lists and arrays- creations, insertion, deletion and traversal

ALORITHM

1. **If T is the empty tree, then the empty list is the preorder, the inorder and the**
2. **Postorder traversal associated with T;**
3. **If T = [N] consists of a single node, the list [N] is the preorder, the inorder and the postorder traversal associated with T;**
4. **Otherwise, T contains a root node n, and subtrees T1,..., Tn: and**
5. **The preorder traversal of the nodes of T is the list containing N, followed, in**
6. **Order by the preorder traversals of T1..., Tn;**
7. **The inorder traversal of the nodes of T is the list containing the inorder**
8. **Traversal of T1 followed by N followed in order by the inorder traversal of each of T2,..., Tn.**

9. The postorder traversal of the nodes of T is the list containing in order the postorder traversal of each of T1,..., Tn, followed by N.

PROGRAM

```
#include<stdlib.h>

#include<iostream.h>

class node

{

public:

int data;

node*Lchild;
```

```
node*Rchild;
};
class bst
{
int item;
node *root;
public: bst();
void insert_node();
void delete_node();
void display_bst();
void preeorder(node*);
void inorder(node*);
void postorder(node*);
};
bst::bst()
{
root=NULL;
}
void bst:: insert_node()
{
node *new_node,*curr,*prev;
new_node=new node;
cout<<"Enter data into new node";
cin>>item;
new_node->data=item;
```

```
new_node->Lchild=NULL;
new_node->Rchild=NULL;
if(root==NULL)
root=new_node;
else
{
curr=prev=root;
while(curr!=NULL)
{
if(new_node->data>curr->data)
{
prev=curr;
curr=curr->Rchild;
}
else
{
prev=curr;
curr=curr->Lchild;
}
}
cout<<"Prev:"<<prev->data<<endl;
if(prev->data>new_node->data)
prev->Lchild=new_node;
else
prev->Rchild=new_node;
```

```
}
}
//code to delete a node
void bst::delete_node()
{
if(root==NULL)
cout<<"Tree is Empty";
else
{
int key;
cout<<"Enter the key value to be deleted";
cin>>key;
node* temp,*parent,*succ_parent;
temp=root;
while(temp!=NULL)
{
if(temp->data==key)
{ //deleting node with two children
if(temp->Lchild!=NULL&&temp->Rchild!=NULL)
{//search for sucessor
node*temp_succ;
temp_succ=temp->Rchild;
while(temp_succ->Lchild!=NULL)
{
succ_parent=temp_succ;
```

```
temp_succ=temp_succ->Lchild;
}
temp->data=temp_succ->data;
succ_parent->Lchild=NULL;
cout<<"Deleted sucess fully";
return;
}
deleting a node having one left child
if(temp->Lchild!=NULL&temp->Rchild==NULL)
{
if(parent->Lchild==temp)
parent->Lchild=temp->Lchild;
else
parent->Rchild=temp->Lchild;
temp=NULL;
delete(temp);
cout<<"Deleted sucess fully";
return;
}
//deleting a node having one right child
if(temp->Lchild==NULL&temp->Rchild!=NULL)
{
if(parent->Lchild==temp)
parent->Lchild=temp->Rchild;
else
```

```
parent->Rchild=temp->Rchild;
temp=NULL;
delete(temp);
cout<<"Deleted sucess fully";
return;
}
//deleting a node having no child
if(temp->Lchild==NULL&temp->Rchild==NULL)
{
if(parent->Lchild==temp)
parent parent->Lchild=NULL;
else
parent->Rchild=NULL;
temp=NULL;
delete(temp);
cout<<"Deleted sucess fully";
return;
}
}
```

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 6

sorting algorithms – bubble, insertion,Quick

AIM

Implementation of sorting algorithms – bubble, insertion, quick,

ALGORITHM

Bubble_Sort (A [ ] , N)

Step 1: Start

Step 2: Take an array of n elements

Step 3: for i=0,…….n-2

Step 4: for j=i+1,…….n-1

Step 5: if arr[j]>arr[j+1] then

Interchange arr[j] and arr[j+1]

End of if

Step 6: Print the sorted array arr

Step 7:Stop

Algorithm:

Selection_Sort (A[ ] , N)

Step 1 : start Begin

Step 2 : Set POS = K

Step 3 : Repeat for J = K + 1 to N –1

Begin

If A[J] < A [POS]

Set POS = J

Step 4 : End For Swap A [K] End For with A [POS]

Step 5 : Stop

Quick Sort

Step 1: Pick an element, called a pivot, from the array.

Step 2: Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.

Step 3: Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 7

searching algorithms – linear search, binary search

AIM

Implementation of searching algorithms – linear search, binary search

ALGORITHM

Algorithm for Linear search

Linear_Search (A[ ], N, val , pos)

Step 1 : Set pos = -1 and k = 0

Step 2 : Repeat while k < N Begin

Step 3 : if A[k] = val

Set pos = k

print pos

Goto step 5

End while

Step 4 : print "Value is not present"

Step 5 : Exit

Binary_Search (A [ ], U_bound, VAL)

Step 1 : set BEG = 0 , END = U_bound , POS = -1

Step 2 : Repeat while (BEG <= END)

Step 3 :set MID = (BEG + END) / 2

POS = MID

print VAL " is available at ", POS

GoTo Step 6

End if

if A [MID] > VAL then

set END = MID – 1

Else

set BEG = MID + 1

End if

End while

Step 5 : if POS = -1 then

print VAL " is not present "

End if

Step 6 : EXIT

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 8

Circular Linked List

AIM

Write a program that uses functions to perform the following operations on circular linked List

(i)Creation (ii) Insertion (iii) Deletion (iv) Traversal

ALGORITHM

Creation

Step 1 - Define a Node structure with two members data and next

Step 2 - Define a Node pointer 'head' and set it to NULL.

Step 3 .Insertion

In a circular linked list, the insertion operation can be performed in three ways. They are as follows...

2.1 Inserting At Beginning of the list

2.2 Inserting At End of the list

2.3 Inserting At Specific location in the list

2.1 Inserting At Beginning of the list

We can use the following steps to insert a new node at beginning of the circular linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set head = newNode and newNode→next = head .

Step 4 - If it is Not Empty then, define a Node pointer 'temp' and initialize with 'head'.

Step 5 - Keep moving the 'temp' to its next node until it reaches to the last node (until 'temp → next == head').

Step 6 - Set 'newNode → next =head', 'head = newNode' and 'temp → next = head'.

2.2 Inserting At End of the list

We can use the following steps to insert a new node at end of the circular linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL).

Step 3 - If it is Empty then, set head = newNode and newNode → next = head.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next == head).

Step 6 - Set temp → next = newNode and newNode → next = head.

2.3 Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the circular linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set head = newNode and newNode → next = head.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

Step 6 - Every time check whether temp is reached to the last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.

Step 7 - If temp is reached to the exact node after which we want to insert the newNode then check whether it is last node (temp → next == head).

Step 8 - If temp is last node then set temp → next = newNode and newNode →

next = head.

Step 9 - If temp is not last node then set newNode → next = temp → next and temp → next = newNode.

3. Deletion

In a circular linked list, the deletion operation can be performed in three ways those are as follows...

3.1 Deleting from Beginning of the list

3.2 Deleting from End of the list

3.3 Deleting a Specific Node

3.1 Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the circular linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize both 'temp1' and 'temp2' with head.

Step 4 - Check whether list is having only one node (temp1 → next == head)

Step 5 - If it is TRUE then set head = NULL and delete temp1 (Setting Empty list conditions)

Step 6 - If it is FALSE move the temp1 until it reaches to the last node.

(until temp1 → next == head)

Step 7 - Then set head = temp2 → next, temp1 → next = head and delete temp2.

3.2 Deleting from End of the list

We can use the following steps to delete a node from end of the circular linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 - Check whether list has only one Node (temp1 → next == head)

Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate from the function. (Setting Empty list condition)

Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until temp1 reaches to the last node in the list. (until temp1 →next == head)

Step 7 - Set temp2 → next = head and delete temp1.

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 9

i) Insertion sort

ii) Merge sort

AIM

Write a program that implements the following

i) Insertion sort

ii) Merge sort

ALGORITHM

I) Insertion Sort:

Step 1: start

Step 2: for i ← 1 to length(A)

Step 3: j ← i

Step 4: while j > 0 and A[j-1] > A[j]

Step 5: swap A[j] and A[j-1]

Step 6: j ← j - 1

Step 7: end while

Step 8: end for

Step 9: stop

II) Merge sort:

Step 1. Divide the data to be sorted in half and put half on each of two tapes.

Step 2. Merge individual pairs of records from the two tapes; write two-record chunks alternately to each of the two output tapes.

Step 3. Merge the two-record chunks from the two output tapes into four-record chunks; write these alternately to the original two input tapes.

Step 4. Merge the four-record chunks into eight-record chunks; write these alternately to the original two output tapes.

Step 5. Repeat until you have one chunk containing all the data, sorted --- that is, for log n passes, where n is the number of records.

Conceptually, merge sort works as follows:

Step 5.1. Divide the unsorted list into two sublists of about half the size.

Step 5.2. Divide each of the two sublists recursively until we have list sizes of length 1, in which case the list itself is returned.

Step 5.3. Merge the two sublists back into one sorted list.

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 10

Dictionary (ADT)using Linked List.

AIM

Write a program to implement all the functions of a dictionary (ADT)using Linked List.

ALGORITHM

Step 1: Create an instance for dictionary

Step 2: initializing dictionary

Step 3: Creation

Step 4: Searching

Step 5:Deletion

Step 6: Stop

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 11

Hashing

AIM

Write a program to implement all the functions of a hashing

ALGORITHM

Step 1: create a hash function

Step 2: insert a node into the hash table

Step 3: find the hash index for the given key.

Step 4: Searching

Step 5: calculated using the hash function

Step 6: Delete

Step 7:Stop

OUTPUT

RESULT AND DISCUSSION

EXPERIMENT – 12

BFS, DFS for each representation

AIM

Implementation of BFS, DFS for each representation

ALGORITHM

BFS

Step 1: Take an Empty Queue.

Step 2: Select a starting node (visiting a node) and insert it into the Queue.

Step 3: Provided that the Queue is not empty, extract the node from the Queue and insert its child nodes (exploring a node) into the Queue.

Step 4: Print the extracted node.

DFS

1. **Start by putting any one of the graph's vertices on top of a stack.**
2. **Take the top item of the stack and add it to the visited list.**
3. **Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.**
4. **Keep repeating steps 2 and 3 until the stack is empty.**

OUTPUT

RESULT AND DISCUSSION