TABLE OF CONTENTS

DATA STRUCTURES LABORATORY
SEMESTER – III

Course Code BCSL305 **CIE Marks** 50
Number of Contact Hours/Week 0:0:2 **SEE Marks** 50
Total Number of Lab Contact Hours 28 **Exam Hours** 03

Course Learning Objectives:

This laboratory course enables students to get practical experience in design, develop, implement, analyze and evaluation/testing of

- Dynamic memory management
- Linear data structures and their applications such as stacks, queues and lists
- Non-Linear data structures and their applications such as trees and graphs

Programs List:

1. Develop a Program in C for the following:

a) Declare a calendar as an array of 7 elements (A dynamically Created array) to represent 7 days of a week. Each Element of the array is a structure having three fields. The first field is the name of the Day (A dynamically allocated String), The second field is the date of the Day (A integer), the third field is the description of the activity for a particular day (A dynamically allocated String).

b) Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

2. Design, Develop and Implement a Program in C for the following operations on Strings. a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP) b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR Support the program with functions for each of the above operations. Don't use Built-in functions.

3. Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations

4. Design, Develop and Implement a Program in C for converting an Infi x Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

5. Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ b. Solving Tower of Hanoi problem with n disks

6. Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX) a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations

7. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo a. Create a SLL of N Students Data by using front insertion . b. Display the status of SLL and count the number of nodes in it c. Perform Insertion / Deletion at End of SLL d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack) e. Exit

8. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo a. Create a DLL of N Employees Data by using end insertion . b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit
9. Design, Develop and Imp lement a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial P(x,y,z) = 6x 2y2z-4yz 5+3x 3yz+2xy 5z-2xyz 3 b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations
10. Design, Develop and Implement a menu driven Program in C for the follow ing operations on Binary Search Tree (BST) of Integers . a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2 b. Traverse the BST in Inorder, Preorder and Post Order c. Search the BST for a given element (KEY) and report the appropriate message d. Exit
11. Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method
12. Given a File of N employee records with a set K of Keys (4 -digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K →L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

Laboratory Outcomes : The student should be able to:

- Analyze various linear and non-linear data structures
- Demonstrate the working nature of different types of data structures and their applications
- Use appropriate searching and sorting algorithms for the give scenario.
- Apply the appropriate data structure for solving real world problems

1.a. C program that declares a calendar as an array of 7 elements, where each element is a structure with dynamically allocated strings for the day name and activity description

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

// Define the structure for a calendar day
struct CalendarDay {
   char* dayName;
   int date;
   char* activity;
};

int main() {
   // Declare an array of 7 CalendarDay structures
   struct CalendarDay calendar[7];

   // Initialize the calendar elements
   for (int i = 0; i < 7; i++) {
      calendar[i].dayName = (char*)malloc(20); // Allocate memory for day name
      calendar[i].activity = (char*)malloc(100); // Allocate memory for activity description

      // You can modify these lines to set specific values for each day
      sprintf(calendar[i].dayName, "Day %d", i + 1);
      calendar[i].date = i + 1;
      sprintf(calendar[i].activity, "Activity for Day %d", i + 1);
   }

   // Print the calendar
   for (int i = 0; i < 7; i++) {
      printf("Day Name: %s\n", calendar[i].dayName);
      printf("Date: %d\n", calendar[i].date);
      printf("Activity: %s\n\n", calendar[i].activity);
   }

   // Free allocated memory
   for (int i = 0; i < 7; i++) {
      free(calendar[i].dayName);
      free(calendar[i].activity);
   }

   return 0;
}
```

(or)

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

// Structure to represent a day in the calendar
struct Day {
   char *dayName;
   int date;
   char *activityDescription;
};

// Function to create a new day in the calendar
void create(struct Day *calendar, int index) {
   printf("Enter details for day %d:\n", index + 1);

   // Dynamically allocate memory for dayName and activityDescription
   calendar[index].dayName = (char *)malloc(sizeof(char) * 20);
   calendar[index].activityDescription = (char *)malloc(sizeof(char) * 100);

   printf("Day Name: ");
   scanf("%s", calendar[index].dayName);

   printf("Date: ");
   scanf("%d", &calendar[index].date);

   printf("Activity Description: ");
   scanf(" %s", calendar[index].activityDescription);
}

// Function to read data from the keyboard for each day in the calendar
void read(struct Day *calendar, int size) {
   for (int i = 0; i < size; ++i) {
      create(calendar, i);
   }
}

// Function to display the weekly activity details report on the screen
void display(struct Day *calendar, int size) {
   printf("\nWeekly Activity Details Report:\n");
   for (int i = 0; i < size; ++i) {
      printf("Day %d: %s, Date: %d, Activity: %s\n", i + 1, calendar[i].dayName, calendar[i].date,
calendar[i].activityDescription);
   }
}
```

```
int main() {
   int numberOfDays;

   printf("Enter the number of days in the week: ");
   scanf("%d", &numberOfDays);

   // Dynamically allocate memory for the calendar array
   struct Day *calendar = (struct Day *)malloc(sizeof(struct Day) * numberOfDays);

   // Read data for each day
   read(calendar, numberOfDays);

   // Display the weekly activity details report
   display(calendar, numberOfDays);

   // Free dynamically allocated memory
   for (int i = 0; i < numberOfDays; ++i) {
      free(calendar[i].dayName);
      free(calendar[i].activityDescription);
   }
   free(calendar);

   return 0;
}
(or)

 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define NUM_DAYS_IN_WEEK 7

// Structure to represent a day
typedef struct
{
   char *acDayName;    // Dynamically allocated string for the day name
   int iDate;        // Date of the day
   char *acActivity;   // Dynamically allocated string for the activity description
}DAYTYPE;

void fnFreeCal(DAYTYPE *);
void fnDispCal(DAYTYPE *);
void fnReadCal(DAYTYPE *);
```

```
DAYTYPE *fnCreateCal();

int main()
{
   // Create the calendar
   DAYTYPE *weeklyCalendar = fnCreateCal();

   // Read data from the keyboard
   fnReadCal(weeklyCalendar);

   // Display the week's activity details
   fnDispCal(weeklyCalendar);

   // Free allocated memory
   fnFreeCal(weeklyCalendar);

   return 0;
}

DAYTYPE *fnCreateCal()
{
   DAYTYPE *calendar = (DAYTYPE *)malloc(NUM_DAYS_IN_WEEK * sizeof(DAYTYPE));


   for(int i = 0; i < NUM_DAYS_IN_WEEK; i++)
           {
      calendar[i].acDayName = NULL;
      calendar[i].iDate = 0;
      calendar[i].acActivity = NULL;
   }

   return calendar;
}

void fnReadCal(DAYTYPE *calendar)
{
           char cChoice;
   for(int i = 0; i < NUM_DAYS_IN_WEEK; i++)
           {
      printf("Do you want to enter details for day %d [Y/N]: ", i + 1);
      scanf("%c", &cChoice); getchar();

      if(tolower(cChoice) == 'n')
           continue;
```

```
        printf("Day Name: ");
        char nameBuffer[50];
        scanf("%s", nameBuffer);
        calendar[i].acDayName = strdup(nameBuffer);  // Dynamically allocate and copy the string


        printf("Date: ");
        scanf("%d", &calendar[i].iDate);


        printf("Activity: ");
        char activityBuffer[100];
        scanf(" %[^\n]", activityBuffer);  // Read the entire line, including spaces
        calendar[i].acActivity = strdup(activityBuffer);

        printf("\n");
        getchar();              //remove trailing enter character in input buffer
    }
}


void fnDispCal(DAYTYPE *calendar)
{
    printf("\nWeek's Activity Details:\n");
    for(int i = 0; i < NUM_DAYS_IN_WEEK; i++)
            {
        printf("Day %d:\n", i + 1);
                    if(calendar[i].iDate == 0)
                    {
                            printf("No Activity\n\n");
                            continue;
                    }

        printf("  Day Name: %s\n", calendar[i].acDayName);
        printf("  Date: %d\n", calendar[i].iDate);
        printf("  Activity: %s\n\n", calendar[i].acActivity);
    }
}

void fnFreeCal(DAYTYPE *calendar)
{
    for(int i = 0; i < NUM_DAYS_IN_WEEK; i++)
            {
        free(calendar[i].acDayName);
        free(calendar[i].acActivity);
```

```
    }
    free(calendar);
}
```

In this program:

1. We define a **CalendarDay** structure with three fields: **dayName**, **date**, and **activity**.
2. We declare an array of 7 **CalendarDay** structures to represent the 7 days of the week.
3. Inside a loop, we dynamically allocate memory for the **dayName** and **activity** fields using **malloc**.
4. We initialize the elements of the calendar with some example data.
5. We print the contents of the calendar.
6. Finally, we free the dynamically allocated memory to avoid memory leaks.

1b. Write functions create(), read() and display(); to create the calendar, to read the data from the keyboard and to print weeks activity details report on screen.

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

// Define the structure for a calendar day
struct CalendarDay {
    char* dayName;
    int date;
    char* activity;
};

// Function to create the calendar
void create(struct CalendarDay calendar[7]) {
    for (int i = 0; i < 7; i++) {
        calendar[i].dayName = (char*)malloc(20); // Allocate memory for day name
        calendar[i].activity = (char*)malloc(100); // Allocate memory for activity description
    }
}

// Function to read data from the keyboard
void read(struct CalendarDay calendar[7]) {
    for (int i = 0; i < 7; i++) {
        printf("Enter day name for Day %d: ", i + 1);
        scanf("%s", calendar[i].dayName);

        printf("Enter date for Day %d: ", i + 1);
        scanf("%d", &calendar[i].date);

        printf("Enter activity for Day %d: ", i + 1);
        scanf(" %[^\n]", calendar[i].activity); // Read the whole line
```

```
    }
}

// Function to display the calendar
void display(struct CalendarDay calendar[7]) {
    printf("\nActivity Details for the Week:\n");
    for (int i = 0; i < 7; i++) {
        printf("Day Name: %s\n", calendar[i].dayName);
        printf("Date: %d\n", calendar[i].date);
        printf("Activity: %s\n\n", calendar[i].activity);
    }
}

int main() {
    struct CalendarDay calendar[7];

    create(calendar);
    read(calendar);
    display(calendar);

    // Free allocated memory
    for (int i = 0; i < 7; i++) {
        free(calendar[i].dayName);
        free(calendar[i].activity);
    }

    return 0;
}
```

- The **create** function allocates memory for the **dayName** and **activity** fields for each day of the week.
- The **read** function reads day names, dates, and activity descriptions for each day from the keyboard.
- The **display** function displays the activity details for the week.

Program2:

Design, Develop and Implement a Program in C for the following operations on Strings
a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR withREP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR
Support the program with functions for each of the above operations. Don't use Built-infunctions.

Theory:

Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**
C language supports a wide range of built-in functions that manipulate null-terminated strings as follows:

strcpy(s1, s2): Copies string s2 into string s1.
strcat(s1, s2): Concatenates string s2 onto the end of string s1.
strlen(s1): Returns the length of string s1.
strcmp(s1, s2): Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
strchr(s1, ch): Returns a pointer to the first occurrence of character ch in string s1.
strstr(s1, s2): Returns a pointer to the first occurrence of string s2 in string s1

Algorithm:

Step 1: Start.
Step 2: Read main string STR, pattern string PAT and replace string REP.
Step 3: Search / find the pattern string PAT in the main string STR.
Step 4: if PAT is found then replace all occurrences of PAT in main string STR with REP string.
Step 5: if PAT is not found give a suitable error message.
Step 6: Stop.

```
#include<stdio.h>

void main()
{
char STR[100],PAT[100],REP[100],ans[100];
int i,j,c,m,k,flag=0;
printf("\nEnter the MAIN string: \n");
gets(STR);
printf("\nEnter a PATTERN string: \n");
gets(PAT);
```

```
printf("\nEnter a REPLACE string: \n");

gets(REP);
i = m = c = j = 0;
while ( STR[c] != '\0')
{
// Checking for Match
if ( STR[m] == PAT[i] )
{
i++;
m++;
flag=1;
if ( PAT[i] == '\0')
{
//copy replace string in ans string

for(k=0; REP[k] != '\0';k++,j++)
ans[j] = REP[k];
i=0;

c=m;
}
}
else //mismatch
{
ans[j] = STR[c];
j++;
c++;
m = c;

i=0;
}
}
if(flag==0)
{

printf("Pattern doesn't found!!!");

}
else
{
ans[j] = '\0';
printf("\nThe RESULTANT string is:%s\n" ,ans);
```

```
}
}
```

Output

Enter the MAIN string:

good morning

Enter a PATTERN string:

morning

Enter a REPLACE string:

evening

The RESULTANT string is: good evening

Or

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

// Function to perform pattern matching and replacement
void findAndReplace(char *mainString, const char *pattern, const char *replace) {
   int mainLen = strlen(mainString);
   int patternLen = strlen(pattern);
   int replaceLen = strlen(replace);

   if (patternLen == 0) {
      printf("Error: Pattern cannot be empty.\n");
      return;
   }

   // Loop through the main string
   for (int i = 0; i <= mainLen - patternLen; ) {
      int j;

      // Check if pattern exists in the current substring
      for (j = 0; j < patternLen; j++) {
         if (mainString[i + j] != pattern[j]) {
            break;
         }
      }

      if (j == patternLen) {
         // Pattern found at position i
         // Replace the pattern with the replacement string
```

```
            for (int k = i; k < i + replaceLen; k++) {
                mainString[k] = replace[k - i];
            }

            // Shift the main string after replacement
            for (int k = i + replaceLen; k <= mainLen - patternLen + replaceLen; k++) {
                mainString[k] = mainString[k + patternLen - replaceLen];
            }

            // Update the length of the main string
            mainLen -= patternLen - replaceLen;
            i += replaceLen;
        } else {
            // No match, move to the next character
            i++;
        }
    }
}

int main() {
    char mainString[1000], pattern[100], replace[100];

    printf("Enter the main string: ");
    scanf(" %[^\n]", mainString);

    printf("Enter the pattern string: ");
    scanf("%s", pattern);

    printf("Enter the replace string: ");
    scanf("%s", replace);

    findAndReplace(mainString, pattern, replace);

    printf("Modified string: %s\n", mainString);

    return 0;
}
```

1. The **findAndReplace** function performs the pattern matching and replacement. It searches for the pattern in the main string and replaces it with the replace string whenever a match is found.
2. The **main** function reads the main string, pattern, and replacement string from the user, and then calls **findAndReplace** to perform the replacement.

3.Develop a menu driven Program in C for the following operations on STACK of Integers
(Array Implementation of Stack with maximum size MAX)
a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate how Stack can be used to check Palindrome
d. Demonstrate Overflow and Underflow situations on Stack
e. Display the status of Stack
f. Exit
Support the program with appropriate functions for each of the above operations

Theory:

Stack is a collection of elements of the similar types. It is also called as last in, first out.
The element inserted first is the last one to be deleted. It is used for various applications like infix to postfix expression, postfix evaluation and for maintaining stack frames for function calling

- **push()** - pushing (storing) an element on the stack.
- **pop()** - removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks;

- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

Algorithm:

Step 1: Start.
Step 2: Initialize stack size MAX and top of stack -1.
Step 3: Push integer element on to stack and display the contents of the stack.
if stack is full give a message as ‘Stack is Overflow’.
Step 3: Pop element from stack along with display the stack contents.
if stack is empty give a message as ‘Stack is Underflow’.
Step 4: Check whether the stack contents are Palindrome or not.
Step 5: Stop

```
#include <stdio.h>
#include <stdlib.h>
int stack[6],rev[6];
int top=-1,k=0;
int size;
void push();void pop();
```

```
void display();
int pali();
void main()
{
int choice,f;
printf("Enter the size for stack\n");
scanf("%d",&size);
printf("1.Push\n2.Pop\n3.Display\n4.Check for Palindrome\n5.Exit\n");
while(1)
{
printf("Enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:push();
break;
case 2:pop();
break;
case 3:display();
break;
case 4:f=pali();
if(f==1)
printf("It's Palindrome\n");
else
printf("It's not a Palindrome\n");
break;
case 5:exit(0);
default:printf("Wrong choice...\n");
}
}
}
void push()
{
int num;
if(top==(size-1))
{
printf("Stack Overflow\n");
}
else
{
printf("Enter the number to be pushed\n");
scanf("%d",&num);top++;
stack[top]=num;
```

```
}
}
void pop()
{
int num;
if(top==-1)
{
printf("Stack Underflow\n");
}
else
{
num=stack[top];
printf("Popped element is %d\n",num);
top--;
}
}
void display()
{
int i;
if(top==-1)
{
printf("Stack Underflow\n");
}
else
{
printf("Stack Contents....\n");
for(i=top;i>=0;i--)
{
printf("%d\n",stack[i]);
rev[k++]=stack[i];
}
}
}
int pali()
{
int i,flag=1;
for(i=top;i>=0;i--)
{
if(stack[i]!=rev[--k])
{

flag=0;
}
```

```
}
return flag;
}
```

Output

```
--------STACK OPERATIONS----------

1.Push

2.Pop

3.Check for Palindrome

4.Display

5.Exit

----------------------

Enter your choice: 1

Enter the element to be inserted: 1

Enter your choice: 1

Enter the element to be inserted: 2

Enter your choice: 1

Enter the element to be inserted: 1

Enter your choice: 1

Enter the element to be inserted: 5

Enter your choice: 2

The poped element: 5

Enter your choice: 4

The stack elements are:

1
2
1
Enter your choice: 3
It's a Palindrome
Enter your choice: 5
```

4. Develop a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

Theory:

Infix: Operators are written in-between their operands. Ex: X + Y

Prefix: Operators are written before their operands. Ex: +X Y

postfix: Operators are written after their operands. Ex: XY+

Algorithm:

Step 1: Read the infix expression as a string.

Step 2: Scan the expression character by character till the end. Repeat the following operations

1. If it is an operand add it to the postfix expression.
2. If it is a left parentheses push it onto the stack.
3. If it is a right parentheses pop out elements from the stack and assign it to the postfix string. Pop out the left parentheses but don't assign to postfix.

Step 3: If it is an operator compare its precedence with that of the element at the top of stack.

1.If it is greater push it onto the stack.

2.Else pop and assign elements in the stack to the postfix expression until you find one such element.

Step 4: If you have reached the end of the expression, pop out any leftover elements in the stack till it becomes empty.

Step 5: Append a null terminator at the end display the result

```
#define size 50 /* size of stack */
#include <ctype.h>
#include <stdio.h>
char s[size];
int top = -1;     /* global declarations */
push(char elem) /* function for push operation */
{
s[++top] = elem;
}
char pop() /* function for pop operation */
{
return (s[top--]);
}
int pr(char elem) /* function for precedence */
{
```

```
switch (elem)
{

case '#':
return 0;
case '(':
return 1 ;
case '+':
case '-':
return 2;
case '*':
case '/':
case '%':
return 3;
case '^':
return 4;
}
 }
void main() /* main program */
{
char infx[50], pofx[50], ch, elem;
int i = 0, k = 0;
printf("\n\nEnter the infix expression ");
scanf("%s", infx);
push('#');
while ((ch = infx[i++]) != '\0')
{
if (ch == '(')
push(ch);
else if (isalnum(ch))
pofx[k++] = ch;
else if (ch == ')')
{
while (s[top] != '(')
{
pofx[k++] = pop();
}
elem = pop(); /* remove ( */
}
else /* operator */
{
while (pr(s[top]) >= pr(ch))
{
```

```
pofx[k++] = pop();
}
push(ch);
}
 }
while (s[top] != '#') /* pop from stack till empty */
{
pofx[k++] = pop();
}

pofx[k] = '\0'; /* make pofx as valid string */
printf("\n\n Given infix expression: %s\n postfix expression is: %s\n", infx, pofx);
}
```

Output

Enter the infix expression (a+b)*c/d^5%1

Given Infix Expn: (a+b)*c/d^5%1

Postfix Expn: ab+c*d5^/1%

5. Develop a Program in C for the following Stack Applications

a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %,^

b. Solving Tower of Hanoi problem with n disks

Algorithm

Step 1: Read the infix expression as a string.

Step 2: Scan the expression character by character till the end. Repeat the following operations

a. If it is an operand push it onto the stack.

b. If it is an operator

1.Pop out two operands from stack

2.Apply the operator onto the popped operands.

3.Store the result back on to the stack

Step 3: On reaching the end of expression pop out the contents of the stack and display as the result.

b. TOWER OF HANOI

The **Tower of Hanoi** is a mathematical game or puzzle. It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

- Only one disk can be moved at a time.

- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2*n* - 1, where *n* is the number of disks

Algorithm:

Step 1:Move a tower of height-1 to an intermediate pole, using the final pole.

Step 2:Move the remaining disk to the final pole.

Step 3:Move the tower of height-1 from the intermediate pole to the final pole using the original pole.

Program 5a:

```
#include<stdio.h>
#include<ctype.h>
float stack[20];

int top=-1;
float eval_postfix(char[]); // function to evalute an given expression
void push(float);         // function to push elements to stack
float pop();      // function to pop the elements from the stack
main()
{
char postfix[20];
float result;
printf("enter postfix expr\n");
scanf("%s", postfix);
result=eval_postfix(postfix);
printf("The result = %f\n",result);
}
float eval_postfix(char postfix[])
{
int i=0,k;
char ch,op1,op2;
float res;
while(postfix[i]!='\0') //check till the end of string
{
ch=postfix[i];
if(isdigit(ch)) //check for digitschecks whether a character is numeric (0-9) or not.
{
k=ch-'0';
push(k);
}
```

```
else
{
op2=pop();
op1=pop();
switch(ch)
{
case '+':push(op1+op2);
break;
case '-':push(op1-op2);
break;
case '*':push(op1*op2);
break;
case '/':push(op1/op2);
break;
case '^':push(pow(op1,op2));
break;
default :printf("illegal\n");
exit(0);
}
}
i++;
}
res=pop();

if(top!=-1)
{
printf("not a valid expression");
exit(1);
}
return(res);
}
void push(float num)
{
top++;
stack[top]=num;
return;
}
float pop()
{
float num;
if(top == -1)
{
printf("not a valid");
```

```
exit(0);
}
else
{
num=stack[top];
top--;
return(num);
}
}
```

OUTPUT

Insert a postfix notation :: 22^32*+

Result :: 10

Program 5b:

```
#include <stdio.h>
void towers(int, char, char, char);
int main()
{
int num;
printf("Enter the number of disks : ");
scanf("%d", &num);
printf("The sequence of moves involved in the Tower of Hanoi are :\n");
towers(num, 'A', 'C', 'B');
return 0;
}
void towers(int num, char frompeg, char topeg, char auxpeg)
{
if (num == 1)
{
printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
return;
}

towers(num - 1, frompeg, auxpeg, topeg);
printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg); towers(num - 1, auxpeg,
topeg, frompeg); }
```

OUTPUT

Enter the number of disks : 3

The sequence of moves involved in the Tower of Hanoi are :

Move disk 1 from peg A to peg C

Move disk 2 from peg A to peg B

Move disk 1 from peg C to peg B

Move disk 3 from peg A to peg C

Move disk 1 from peg B to peg A

Move disk 2 from peg B to peg C

Move disk 1 from peg A to peg C

6.Develop a menu driven Program in C for the following operations on circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
Support the program with appropriate functions for each of the above operations

Theory:

Circular queue is a linear data structure. It follows FIFO principle. In **circular queue** the last node is connected back to the first node to make a **circle**.It is also called FIFO structure. Elements are added at the rear end and the elements are deleted at front end of the **queue**. The queue is considered as a circular queue when the positions 0 and MAX-1 are adjacent.

ALGORITHM:

Step 1: Start.
Step 2: Initialize queue size to MAX.
Step 3: Insert the elements into circular queue. If queue is full give a message as ‘queue is overflow”
Step 4: Delete an element from the circular queue. If queue is empty give a message as ‘queue is underflow’.
Step 5: Display the contents of the queue.
Step 6: Stop.

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

```
#include <conio.h>
#include <string.h>
#define SIZE 5
char CQ[SIZE];
int front=-1, rear=-1;
int ch;
void CQ_Insert();
void CQ_Delet();
void CQ_Display();
void main()
{
printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
while(1)
{
printf("Enter your choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: CQ_Insert();
break;
case 2:CQ_Delet();
break;
case 3:CQ_Display();
break;
case 4: exit(0);
}
}
}
void CQ_Insert()
{
char ele;
if(front==(rear+1)%SIZE)
{
printf("Circular Queue Full\n");
return;
}
if(front==-1)
front++;
printf("Enter the element to be inserted\n");
scanf("\n%c",&ele);
rear = (rear+1)%SIZE;
CQ[rear] =ele;
}
```

```
void CQ_Delet()
{
char item;
if(front == -1)
{
printf("Circular Queue Empty\n");
return;
}
else if(front == rear)
{
item=CQ[front];
printf("Deleted element is: %c\n",item);
front=-1;
rear=-1;
}
else
{
item =CQ[front];
printf("Deleted element is: %c\n",item);
front = (front+1)%SIZE;
}
}
void CQ_Display()
{
int i;
if(front==-1)
printf("Circular Queue is Empty\n");
else
{
printf("Elements of the circular queue are..\n");
for(i=front;i!=rear;i=(i+1)%SIZE)
{
printf("%c\t",CQ[i]);
}
printf("%c\n",CQ[i]);
}
}
```

Output

Circular Queue operations

1.insert

2.delete

3.display

4.exit

Enter your choice:1

Enter element to be insert: a

Enter your choice:1

Enter element to be insert: b

Enter your choice:1

Enter element to be insert: c

Enter your choice:3

a b c

Enter your choice:2

Deleted element is: a

Enter your choice:3

b c

Enter your choice:4

7.Develop a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem,PhNo*

a. Create a SLL of N Students Data by using *front insertion*.

b. Display the status of SLL and count the number of nodes in it

c. Perform Insertion / Deletion at End of SLL

d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)

e. Exit

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int count=0;
struct node
{
int sem,phno;
```

```
char name[20],branch[10],usn[20];
struct node *next;
}*first=NULL,*last=NULL,*temp=NULL, *temp1;
void create()
{
int sem,phno;
char name[20],branch[10],usn[20]; temp=(struct node*)malloc(sizeof(struct node));
printf("\n Enter usn,name, branch, sem, phno of student : ");
scanf("%s %s %s %d %d", usn, name,branch, &sem,&phno);
strcpy(temp->usn,usn);
strcpy(temp->name,name);
strcpy(temp->branch,branch);
temp->sem = sem;
temp->phno = phno;
temp->next=NULL;
count++;
}
void insert_atfirst()
{
if (first == NULL)
{
create();
first = temp; last = first;
}
else
{
create();
temp->next = first; first = temp;
}
}
void insert_atlast()
{
if(first==NULL)
{
create(); first = temp; last = first;
}
else
{
```

```
create();
last->next = temp;
last = temp;
}
}
void display()
{
temp1=first; if(temp1 ==NULL)
{
printf("List empty to display \n");
return;
}
printf("\n Linked list elements from begining : \n");
while (temp1!= NULL)
{
printf("%s %s %s %d %d\n", temp1->usn, temp1->name,temp1->branch,temp1->sem,temp1->phno
);
temp1 = temp1->next;}
printf(" No of students = %d ", count);
}
int deleteend()
{
struct node *temp; temp=first;
if(temp->next==NULL)
{
free(temp);
first=NULL;
}
else
{
while(temp->next!=last)
temp=temp->next;
printf("%s %s %s %d %d\n", last->usn, last->name,last->branch, last->sem, last->phno );
free(last); temp->next=NULL;
last=temp;
}
count--;
```

```
return 0;
}
int deletefront()
{
struct node *temp; temp=first;
if(temp->next==NULL)
{
free(temp);
first=NULL;
return 0;
}
else
{
first=temp->next;
printf("%s %s %s %d %d", temp->usn, temp->name,temp->branch,temp->sem, temp->phno );
free(temp);
}
count--;
return 0;
}
void main()
{
int ch,n,i;
first=NULL;
temp = temp1 = NULL; printf("-----------------MENU----------------------\n");
printf("\n 1 â€“ create a SLL of n emp");
printf("\n 2 - Display from beginning");
printf("\n 3 - Insert at end");
printf("\n 4 - delete at end");
printf("\n 5 - Insert at beg");
printf("\n 6 - delete at beg");
printf("\n 7 - exit\n");
printf("-------------------------------------------\n");
while (1)
{
printf("\n Enter choice : ");
scanf("%d", &ch);
switch (ch)
```

```
{
case 1:
printf("\n Enter no of students : ");
scanf("%d", &n);
for(i=0;i<n;i++)
insert_atfirst(); break;
case 2: display();break;
case 3:insert_atlast(); break;
case 4:deleteend();break;
case 5: insert_atfirst(); break;
case 6: deletefront(); break;
case 7: exit(0);
default: printf("wrong choice\n");
}
}
}
```

Output

```
linux:~/dslab #gedit slink.c
linux:~/dslab #gcc slink.c
linux:~/dslab # ./a.out
–        ----------MENU----------
1        –    create a SLL of n emp
2        -Display from beginning
3        -Insert at end
4        -delete at end
5        -Insert at beg
6        -delete at beg
7        -
------------------------------------------------
Enter choice : 1
Enter no of students : 2
Enter usn,name, branch, sem, phno of student : 007 vijay CSE 3 121
Enter usn,name, branch, sem, phno of student : 100 yashas CSE 3 911
Enter choice : 2
Linked list elements from begining :
100 yashas CSE 3 911
007 vijay CSE 3 121
```

No of students = 2

Enter choice : 3
Enter usn,name, branch, sem, phno of student : 001 raj CSE 3 111

Enter choice : 2

Linked list elements from
begining : 100 yashas CSE 3 911
007 vijay CSE 3 121

001 raj CSE 3 111
No of students = 3

Enter choice : 4
001 raj CSE 3
111 Enter
choice : 2

Linked list elements from
begining :
100 yashas CSE 3 911

007 vijay CSE 3 121
No of students = 2

Enter choice : 5
Enter usn,name, branch, sem, phno of student : 003 harsh cse 3 111
Enter choice : 2

Linked list elements from begining :
003 harsh cse 3 111
100 yashas CSE 3 911

007 vijay CSE 3 121
No of students = 3

Enter choice : 6 003 harsh cse 3 111
Enter choice : 2

Linked list elements from
begining :
100 yashas CSE 3 911

007 vijay CSE 3 121
No of students = 2
Enter choice : 7

8.Develop a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo*

a. Create a DLL of N Employees Data by using *end insertion*.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue.
f. Exit

```
#include<string.
h> int count=0;
struct node
{
struct node
*prev; int
ssn,phno; float
sal;
char name[20],dept[10],desg[20];
struct node *next;
}*h,*temp,*temp1,*temp2,*temp
4;

void create()
{
int
ssn,phno;
float sal;
char name[20],dept[10],desg[20];
temp =(struct node *)malloc(sizeof(struct node));
temp->prev = NULL;
temp->next = NULL;
printf("\n Enter ssn,name,department, designation, salary and phno of employee :
"); scanf("%d %s %s %s %f %d", &ssn, name,dept,desg,&sal, &phno);
temp->ssn = ssn;
strcpy(temp->name,name);
strcpy(temp->dept,dept);
strcpy(temp->desg,desg);
 temp->sal = sal;
temp->phno = phno;
count++;}
```

```
void insertbeg()
{
if (h == NULL)
{
create(
); h =
temp;
temp1 = h;
}
else
{
create(); temp->next =
h; h->prev = temp;
h = temp;
}
}
void insertend()
{
if(h==NULL)
{
create(
); h =
temp;
temp1 = h;
}
else
{
create(); temp1->next =
temp; temp->prev = temp1;
temp1 = temp;
}
}
void displaybeg()
{
temp2 =h;
if(temp2 == NULL)
{
printf("List empty to display
```

```
\n"); return;}
printf("\n Linked list elements from begining : \n");
while (temp2!= NULL)
{
printf("%d %s %s %s %f %d\n", temp2->ssn, temp2->name,temp2->dept,temp2->desg,temp2->sal,
 temp2->phno );
temp2 = temp2->next;
}
printf(" No of employees = %d ", count);
}
int deleteend()
{
struct node *temp; temp=h; if(temp-
>next==NULL)
{
free(temp);
h=NUL
L;
return
0;
}
else
{
temp2=temp1->prev; temp2->next=NULL;
printf("%d %s %s %s %f %d\n", temp1->ssn, temp1->name,temp1->dept, temp1->desg,temp1-
>sal,
temp1->phno );
free(temp1);
}
count
--;
return
0;
}
int deletebeg()
{
struct node *temp; temp=h; if(temp-
>next==NULL)
{
```

```
free(temp);
h=NULL;}
else
{
h=h->next;
printf("%d %s %s %s %f %d", temp->ssn, temp->name,temp->dept, temp->desg,temp->sal, temp->phno );
free(temp);
}
count
--;
return
0;
}
void main()
{
int ch,n,i; h=NULL;
temp = temp1 = NULL;
printf("-----------------MENU--------------------\n");
printf("\n 1 â€“ create a DLL of n emp");
printf("\n 2 - Display from beginning");
printf("\n 3 - Insert at end");
printf("\n 4 - delete at end");
printf("\n 5 - Insert at beg");
printf("\n 6 - delete at beg");
printf("\n 7 - exit\n");
printf("-----------------------------------------\n");
while (1)
{
printf("\n Enter choice : "); scanf("%d", &ch);
switch (ch)
{
case 1:
printf("\n Enter no of employees : "); scanf("%d", &n);
for(i=0;i<n;i++)
insertend();
break;
case 2:
displaybeg();
```

```
break;
case 3: insertend(); break;
case 4:
deleteend();
break;
case 5: insertbeg();
break;
case 6:
deletebeg();
break;
case 7: exit(0);
default: printf("wrong choice\n");
}
}
}
```

Output

```
linux:~/dslab #gedit
dlink.c linux:~/dslab #gcc
dlink.c linux:~/dslab #
./a.out
                MENU----------
1 –Create a DLL of n emp
2 - Display from beginning
3 - Insert at end
4 - Delete at end
5 - Insert at beg
6 - Delete at beg
7 - exit
----------------------------------------
Enter choice : 1
Enter no of employees : 2
Enter ssn,name,department, designation, salary and phno of employee :
1 RAJ SALES MANAGER 15000 911
Enter ssn,name,department, designation, salary and phno of employee :
2 RAVI HR ASST 10000 123
Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
```

2 RAVI HR ASST 10000.000000
123 No of employees = 2
Enter choice : 3
Enter ssn,name,department, designation, salary and phno of
employee : 3 RAM MARKET MANAGER 50000 111
Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123
3 RAM MARKET MANAGER 50000.000000
111 No of employees = 3
Enter choice : 4
3 RAM MARKET MANAGER 50000.000000
111 Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000
123 No of employees = 2
Enter choice : 5
Enter ssn,name,department, designation, salary and phno of
employee : 0 ALEX EXE TRAINEE 2000 133
Enter choice : 2
Linked list elements from begining :
0 ALEX EXE TRAINEE 2000.000000 133
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000
123 No of employees = 3
Enter choice : 6
0 ALEX EXE TRAINEE 2000.000000 133
Enter choice : 2
Linked list elements from begining :
1 RAJ SALES MANAGER 15000.000000 911
2 RAVI HR ASST 10000.000000 123
No of employees = 2
Enter choice : 7
Exit

9.Develop a Program in C for the following operationson Singly Circular Linked List (SCLL)with header nodes

a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3 b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

Support the program with appropriate functions for each of the above operations

Circular Linked List is a variation of Linked list in which first element points to last element and last element points to first element. Both Singly Linked List and Doubly Linked List can be made into as circular linked list.

Adding two polynomials are represented using linked lists

Representation of a Polynomial: A polynomial is an expression that contains more than two terms. A term is made up of coefficient and exponent.

An example of polynomial is

$P(x) = 4x^3+6x^2+7x+9$

A polynomial thus may be represented using arrays or linked lists. Array representation assumes that the exponents of the given expression are arranged from 0 to the highest value (degree), which is represented by the subscript of the array beginning with 0. The coefficients of the respective exponent are placed at an appropriate index in the array. The array representation for the above polynomial expression is given below:

A polynomial may also be represented using a linked list. A structure may be defined such that it contains two parts- one is the coefficient and second is the corresponding exponent. The structure definition may be given as shown below:

```
struct polynomial
{
        int coefficient;
        int exponent;
        struct polynomial *next;
};
```

Addition of two Polynomials:

For adding two polynomials using arrays is straightforward method, since both the arrays may be added up element wise beginning from 0 to n-1, resulting in addition of two polynomials. Addition of two polynomials using linked list requires comparing the exponents, and wherever the exponents are found to be same, the coefficients are added up. For terms with different exponents, the complete term is simply added to the result thereby making it a part of addition result. The complete program to add two polynomials is given in subsequent section.

```
#include<stdio.h>
#include<alloc.h>
#include<math.h>
struct node
{
```

```
int cf, px, py, pz;
int flag;
struct node *link;
};
typedef struct node NODE;
NODE* getnode()
{
NODE *x;
x=(NODE*)malloc(sizeof(NODE));
if(x==NULL)
{
printf("Insufficient memory\n");
exit(0);
}
return x;
}
void display(NODE *head)
{
NODE *temp;
if(head->link==head)
{
printf("Polynomial does not exist\n");
return;
}
temp=head->link;
printf("\n");
while(temp!=head)
{
printf("%d x^%d y^%d z^%d",temp->cf,temp->px,temp->py,temp-
>pz);
if(temp->link != head)
printf(" + ");
temp=temp->link;
}
printf("\n");
}
NODE* insert_rear(int cf,int x,int y,int z,NODE *head)
{
```

```
NODE *temp,*cur;
temp=getnode();
temp->cf=cf;
temp->px=x;
temp->py=y;
temp->pz=z;
cur=head->link;
while(cur->link!=head)
{
cur=cur->link;
}
cur->link=temp;
temp->link=head;
return head;
}
NODE* read_poly(NODE *head)
{
int px, py, pz, cf, ch;
printf("\nEnter coeff: ");
scanf("%d",&cf);
printf("\nEnter x, y, z powers(0-indiacate NO term): ");
scanf("%d%d%d", &px, &py, &pz);
head=insert_rear(cf,px,py,pz,head);
printf("\nIf you wish to continue press 1 otherwise 0: ");
scanf("%d", &ch);
while(ch != 0)
{
printf("\nEnter coeff: ");
scanf("%d",&cf);
printf("\nEnter x, y, z powers(0-indiacate NO term): ");
scanf("%d%d%d", &px, &py, &pz);
head=insert_rear(cf,px,py,pz,head);
printf("\nIf you wish to continue press 1 otherwise 0: ");
scanf("%d", &ch);
}
return head;
}
NODE* add_poly(NODE *h1,NODE *h2,NODE *h3)
```

```
{
NODE *p1,*p2;
int x1,x2,y1,y2,z1,z2,cf1,cf2,cf;
p1=h1->link;
while(p1!=h1)
{
x1=p1->px;
y1=p1->py;
z1=p1->pz;
cf1=p1->cf;
p2=h2->link;
while(p2!=h2)
{
x2=p2->px;
y2=p2->py;
z2=p2->pz;
cf2=p2->cf;
if(x1==x2 && y1==y2 && z1==z2)break;
p2=p2->link;
}
if(p2!=h2)
{
cf=cf1+cf2;
p2->flag=1;
if(cf!=0)
h3=insert_rear(cf,x1,y1,z1,h3);
}
else
h3=insert_rear(cf1,x1,y1,z1,h3);
p1=p1->link;
}
p2=h2->link;
while(p2!=h2)
{
if(p2->flag==0)
h3=insert_rear(p2->cf,p2->px,p2->py,p2->pz,h3);
p2=p2->link;
}
```

```
return h3;
}
void evaluate(NODE *h1)
{
NODE *head;
int x, y, z;
float result=0.0;
head=h1;
printf("\nEnter x, y, z, terms to evaluate:\n");
scanf("%d%d%d", &x, &y, &z);
while(h1->link != head)
{
result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) *
pow(z,h1->pz));
h1=h1->link;
}
result = result + (h1->cf * pow(x,h1->px) * pow(y,h1->py) *
pow(z,h1->pz));
printf("\nPolynomial result is: %f", result);
}
void main()
{
NODE *h1,*h2,*h3;
int ch;
clrscr();
h1=getnode();
h2=getnode();
h3=getnode();
h1->link=h1;
h2->link=h2;
h3->link=h3;
while(1)
{
printf("\n\n1.Evaluate polynomial\n2.Add two
polynomials\n3.Exit\n");
printf("Enter your choice: ");
scanf("%d", &ch);
switch(ch)
```

```
{
case 1:
printf("\nEnter polynomial to evaluate:\n");
h1=read_poly(h1);
display(h1);
evaluate(h1);
break;
case 2:
printf("\nEnter the first polynomial:");
h1=read_poly(h1);
printf("\nEnter the second polynomial:");
h2=read_poly(h2);
h3=add_poly(h1,h2,h3);
printf("\nFirst polynomial is: ");
display(h1);
printf("\nSecond polynomial is: ");
display(h2);
printf("\nThe sum of 2 polynomials is: ");
display(h3);
break;
case 3:
exit(0); break;
default:
printf("\nInvalid entry");
break;
}
getch();
}
}
```

Output:

1. Evaluate polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
2. Add two polynomials
3. Exit

Enter your choice: 1
Enter polynomial to evaluate:
Enter coeff: 6 Enter x, y, z powers (0-indiacate NO term: 2 2 1
If you wish to continue press 1 otherwise 0: 1

Enter coeff: -4 Enter x, y, z powers (0-indiacate NO term: 0 1 5
If you wish to continue press 1 otherwise 0: 1
Enter coeff: 3
Enter x, y, z powers (0-indiacate NO term: 3 1 1
If you wish to continue press 1 otherwise 0: 1
Enter coeff: 2
x, y, z powers (0-indiacate NO term: 1 5 1
If you wish to continue press 1 otherwise 0: 1
Enter coeff: -2
Enter x, y, z powers (0-indiacate NO term: 1 1 3
If you wish to continue press 1 otherwise 0: 0
Polynomial is: 6 x^2 y^2 z^1 + -4 x^0 y^1 z^5 + 3 x^3 y^1 z^1 + 2 x^1 y^5 z^1 + -2 x^1 y^1 z^3
Enter x, y, z, terms to evaluate: 1 1 1
Polynomial result is: 5.000000
1. Evaluate polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
2. Add two polynomials
3. Exit Enter your choice: 2
Enter 1st polynomial: Enter coeff: 4
Enter x, y, z powers (0-indiacate NO term: 2 2 2
If you wish to continue press 1 otherwise 0: 1
Enter coeff: 3 Enter x, y, z powers (0-indiacate NO term: 1 1 2
If you wish to continue press 1 otherwise 0: 0
Enter 2nd polynomial: Enter coeff: 6
Enter x, y, z powers (0-indiacate NO term: 2 2 2
If you wish to continue press 1 otherwise 0: 0
Polynomial is: 1st Polynomial is: 4 x^2 y^2 z^2 + 3 x^1 y^1 z^2
2nd Polynomial is: 6 x^2 y^2 z^2 Added polynomial: 10 x^2 y^2 z^2 + 3 x^1 y^1 z^2 1.
Evaluate polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
2. Add two polynomials 3. Exit Enter your choice: 3

10.Develop a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers .

a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2

b. Traverse the BST in Inorder, Preorder and Post Order

c. Search the BST for a given element (KEY) and report the appropriate message

d. Exit

A binary search tree (BST) is a tree in which all nodes follows the below mentioned properties −

- The left sub-tree of a node has key less than or equal to its parent node's key.
- The right sub-tree of a node has key greater than or equal to its parent node's key.

Thus, a binary search tree (BST) divides all its sub-trees into two segments; *left* sub-tree and *right* sub-tree and can be defined as −

left_subtree (keys) ≤ node (key) ≤ right_subtree (keys)

Basic Operations

Following are basic primary operations of a tree which are following.

- **Search** − search an element in a tree.
- **Insert** − insert an element in a tree.
- **Preorder Traversal** − traverse a tree in a preorder manner.
- **Inorder Traversal** − traverse a tree in an inorder manner.
- **Postorder Traversal** − traverse a tree in a postorder manner.

```
#include <stdio.h>
#include <stdlib.h>
struct BST
{
int data;
struct BST *left;
struct BST *right;
};
typedef struct BST NODE;
NODE *node;
NODE* createtree(NODE *node, int data)
{
if (node == NULL)
{
NODE *temp;
temp= (NODE*)malloc(sizeof(NODE));
temp->data = data;
temp->left = temp->right = NULL;
return temp;
}
if (data < (node->data))
{
```

```
node->left = createtree(node->left, data);
}
else if (data > node->data)
{
node -> right = createtree(node->right, data);
}
return node;
}
NODE* search(NODE *node, int data)
{
if(node == NULL)
printf("\nElement not found");
else if(data < node->data)
{
node->left=search(node->left, data);
}
else if(data > node->data)
{
node->right=search(node->right, data);
}
else
printf("\nElement found is: %d", node->data);
return node;
}
void inorder(NODE *node)
{
if(node != NULL)
{
inorder(node->left);
printf("%d\t", node->data);
inorder(node->right);
}
}
void preorder(NODE *node)
{
if(node != NULL)
{
printf("%d\t", node->data);
```

```
preorder(node->left);
preorder(node->right);
}
}
void postorder(NODE *node)
{
if(node != NULL)
{
postorder(node->left);
postorder(node->right);
printf("%d\t", node->data);
}
}
NODE* findMin(NODE *node)
{
if(node==NULL)
{
return NULL;
}
if(node->left)
return findMin(node->left);
else
return node;
}
void main()
{
int data, ch, i, n;
NODE *root=NULL;
clrscr();
while (1)
{
printf("\n1.Insertion in Binary Search Tree");
printf("\n2.Search Element in Binary Search Tree");
printf("\n3.Inorder\n4.Preorder\n5.Postorder\n6.Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch (ch)
{
```

```
case 1: printf("\nEnter N value: " );
scanf("%d", &n);
printf("\nEnter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)\n");
for(i=0; i<n; i++)
{
scanf("%d", &data);
root=createtree(root, data);
}
break;
case 2: printf("\nEnter the element to search: ");
scanf("%d", &data);
root=search(root, data);
break;
case 3: printf("\nInorder Traversal: \n");
inorder(root);
break;
case 4: printf("\nPreorder Traversal: \n");
preorder(root);
break;
case 5: printf("\nPostorder Traversal: \n");
postorder(root);
break;
case 6: exit(0);
default: printf("\nWrong option");
break;
}
}
}
```

Output:

```
1. Insertion in Binary Search Tree
 2. Search Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder
6. Exit
Enter your choice: 1
Enter N value: 12 Enter the values to create BST like(6,9,5,2,8,15,24,14,7,8,5,2)
6 9 5 2 8 15 24 14 7 8 5 2
```

1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder
6. Exit
Enter your choice: 3
Inorder Traversal: 2 5 6 7 8 9 14 15 24
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder
6. Exit
Enter your choice: 4
Preorder Traversal: 6 5 2 9 8 7 15 14 24
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder
6. Exit
Enter your choice: 5
Postorder Traversal: 2 5 7 8 14 24 15 9 6
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder
6. Exit
Enter your choice: 2
Enter the element to search: 24
Element found is: 24
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder

6. Exit
Enter your choice: 2
Enter the element to search: 50
Element not found
1. Insertion in Binary Search Tree
2. Search Element in Binary Search Tree
3. Inorder
4. Preorder
5. Postorder
6. Exit Enter your choice: 6

11.Develop a Program in C for the following operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

Depth-first search (DFS)

There are various ways to traverse (visit all the nodes) of a graph systematically. A couple of these ways (depth-first and breadth-first) give us some information about graph structure (e.g. connectedness).

In depth-first search the idea is to travel as deep as possible from neighbour to neighbour before backtracking. What determines how deep is possible is that you must follow edges, and you don't visit any vertex twice.

To do this properly we need to keep track of which vertices have already been visited, plus how we got to (the path to...) where we currently are, so that we can backtrack. We could keep track of which nodes were visited in a boolean array, and a stack to push nodes onto that we mean to visit (the course Readings have a recursive algorithm for DFS which takes a slightly different approach).

```
#include<stdio.h>
#include<conio.h>
int a[10][10], n, m, i, j, source, s[10], b[10];
int visited[10];
void create()
{
printf("\nEnter the number of vertices of the digraph: ");
scanf("%d", &n);
printf("\nEnter the adjacency matrix of the graph:\n");
for(i=1; i<=n; i++)
for(j=1; j<=n; j++)
scanf("%d", &a[i][j]);
```

```
}
void bfs()
{
int q[10], u, front=0, rear=-1;
printf("\nEnter the source vertex to find other nodes reachable or not: ");
scanf("%d", &source);
q[++rear] = source;
visited[source] = 1;
printf("\nThe reachable vertices are: ");
while(front<=rear)
{
u = q[front++];
for(i=1; i<=n; i++)
{
if(a[u][i] == 1 && visited[i] == 0)
{
q[++rear] = i;
visited[i] = 1;
printf("\n%d", i);
}
}
}
}
void dfs(int source)
{
int v, top = -1;
s[++top] = 1;
b[source] = 1;
for(v=1; v<=n; v++)
{
if(a[source][v] == 1 && b[v] == 0)
{
printf("\n%d -> %d", source, v);
dfs(v);
}
}
}
void main()
```

```
{
int ch;
clrscr();
while(1)
{
printf("\n1.Create Graph\n2.BFS\n3.Check graph connected or not(DFS)\n4.Exit");
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1: create();
         break;
case 2: bfs();
        for(i=1;i<=n;i++)
        if(visited[i]==0)
        printf("\nThe vertex that is not rechable %d" ,i);
        break;
case 3: printf("\nEnter the source vertex to find the connectivity: ");
        scanf("%d",&source);
        m=1;
        dfs(source);
        for(i=1;i<=n;i++) {
        if(b[i]==0)
        m=0;
        }
        if(m==1)
        printf("\nGraph is Connected");
        else
        printf("\nGraph is not Connected");
        break;
default: exit(0);
}
}
}
```

OUTPUT:

1. Create Graph
2.BFS

3.Check graph connected or not (DFS)
4.Exit
Enter your choice: 1
Enter the number of vertices of the digraph: 5
Enter the adjacency matrix of the graph:
0 1 1 1 0
1 0 0 1 1
1 0 0 1 0
1 1 1 0 1
0 1 0 1 0

1. Create Graph
2.BFS
3.Check graph connected or not (DFS)
4.Exit
Enter your choice: 2
Enter the source vertex to find other nodes reachable or not: 1
2 3 4 5
1. Create Graph
2.BFS
3.Check graph connected or not (DFS)
4.Exit
Enter your choice: 3
Enter the source vertex to find the connectivity: 1
1 -> 2
2 -> 4
4 -> 3
4->5
Graph is Connected
1. Create Graph 2.BFS 3.Check graph connected or not (DFS) 4.Exit Enter your choice: 4

12. Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and

addresses in L are Integers. Develop a Program in C that uses Hash function H: K →L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

Direct-address table

If the keys are drawn from the reasoning small universe U = {0, 1, . . . , m-1} of keys, a solution is to use a Table T[0, . m-1], indexed by keys. To represent the dynamic set, we use an array, or direct-address table, denoted by T[0 . . m-1], in which each slot corresponds to a key in the universe.

Following figure illustrates the approach.

Each key in the universe U i.e., Collection, corresponds to an index in the table T[0 . . m-1]. Using this approach, all three basic operations (dictionary operations) take θ(1) in the worst case.

Hash Tables

When the size of the universe is much larger the same approach (direct address table) could still work in principle, but the size of the table would make it impractical. A solution is to map the keys onto a small range, using a function called a hash function. The resulting data structure is called hash table.

With direct addressing, an element with key k is stored in slot k. With hashing =, this same element is stored in slot h(k); that is we use a hash function h to compute the slot from the key. Hash function maps the universe U of keys into the slot of a hash table T[0 . . .m-1].

h: $U \rightarrow$ {0, 1, . . ., m-1}

More formally, suppose we want to store a set of size *n* in a table of size *m*. The ratio α = *n/m* is called a load factor, that is, the average number of elements stored in a Table. Assume we have a hash function h that maps each key *k*□*U* to an integer name h(*k*) □[0 . . *m*-1]. The basic idea is to store key k in location T[h(k)].

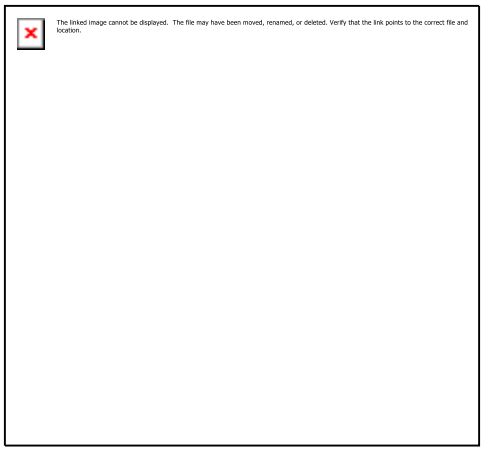
Typical, hash functions generate "random looking" valves. For example, the following function usually works well

h(*k*) = *k* mod *m* where *m* is a prime number.

Is there any point of the hash function? Yes, the point of the hash function is to reduce the range of array indices that need to be handled.

Collision

As keys are inserted in the table, it is possible that two keys may hash to the same table slot. If the hash function distributes the elements uniformly over the table, the number of conclusions cannot be too large on the average, but the birthday paradox makes it very likely that there will be at least one collision, even for a lightly loaded table



A hash function h map the keys *k* and *j* to the same slot, so they collide.

There are two basic methods for handling collisions in a hash table: Chaining and Open addressing.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
 int create(int);
void linear_prob(int[], int, int);
 void display (int[]);
void main()
```

```
{
int
a[MAX],num,key,i;
int ans=1;
printf(" collision handling by linear probing : \n");
 for (i=0;i<MAX;i++)
{
a[i] = -1;
}
do
{
printf("\n Enter the data");
scanf("%4d", &num);
key=create(num);
linear_prob(a,key,num);
printf("\n Do you wish to continue ? (1/0) ");
scanf("%d",&ans);
}while(ans);
display(a);
}
int create(int num)
{
int key; key=num%100;
return key;}
void linear_prob(int a[MAX], int key, int num)
{
int flag, i,
count=0;
flag=0;
if(a[key]== -1)
{
a[key] = num;
}
else
{
printf("\nCollision Detected...!!!\n");
i=0;
while(i<MAX)
```

```
{
if (a[i]!=-1)
count++; i++;
}
printf("Collision avoided successfully using LINEAR PROBING\n");
 if(count == MAX)
{
printf("\n Hash table is full");
display(a);
exit(1);
}
for(i=key+1; i<MAX; i++)
 if(a[i] == -1)
{
a[i] = num; flag =1;
break;
}
//for(i=0;i<key;i++)
i=0;
while((i<key) && (flag==0))
{
if(a[i] == -1)
{
a[i] =
num;
flag=1;
break;
}
i++;
}
}
}
void display(int a[MAX])
{
int i,choice;
printf("1.Display ALL\n 2.Filtered Display\n");
scanf("%d",&choice);
if(choice==1)
```

```
{
printf("\n the hash table is\n");
 for(i=0; i<MAX; i++)
printf("\n %d %d ", i, a[i]);
}
else
{
printf("\n the hash table
is\n"); for(i=0; i<MAX;
i++) if(a[i]!=-1)
{
printf("\n %d %d ", i, a[i]);
continue;
}}}
```

Output

```
linux:~/dslab #gedit
hash.c linux:~/dslab
#gcc hash.c
linux:~/dslab # ./a.out
collision handling by linear probing :
Enter the data1234
Do you wish to continue ? (1/0) 1

Enter the data2548

Do you wish to continue ? (1/0) 1

Enter the data3256

Do you wish to continue ? (1/0) 1

Enter the data1299

Do you wish to continue ? (1/0) 1

Enter the data1298

Do you wish to continue ? (1/0) 1
```

Enter the data1398

Collision Detected...!!!
Collision avoided successfully using LINEAR PROBING

Do you wish to continue ?
(1/0) 0 1.Display ALL
2.Filtered
Display 2

the hash table is

0 1398
34 1234
48 2548
56 3256
98 1298
99 1299
100

13.Viva Questions

1) What is data structure?

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

2) Differentiate file structure from storage structure.

Basically, the key difference 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 search 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 of data storage.

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

To do this, an indexed loop is used, such that the counter runs from 0 to the array size minus one. In this manner, we are able to reference all the elements in sequence by using the loop counter as the array subscript.

6) In what areas do data structures applied?

Data structures is important 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 short for Last In First Out, and refers to 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 structures that can simulates 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 actually an extension of the linked list structures.

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

Recursion, which is basically 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 a 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 actually 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?

Aside 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 is short 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 takes a divide-and-conquer approach to sorting 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 continuous until you have one single sorted list.

19) Differentiate NULL and VOID.

Null is actually a value, whereas Void is a data type identifier. A variable that is given a Null value simply indicates an empty value. 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 a very ideal data structure because it can be modified easily. This means that modifying 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. Using this method, each element in the list is checked and compared against the target key, and is repeated until found or if the end of the list 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?

Basically, 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, 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. To perform this, 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 with 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 (unsigned 8 bit number has a range 0-255, while 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 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 actually 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 intended 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.

Data that is stored in a stack follows a LIFO pattern. This means that data access follows a sequence wherein the last data to be stored will 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.

1. 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 target is not in the root item, check if target is smaller than the root's value
5. if 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, with each node being linked in a sequential manner.

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 non linear data structure.

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, non-linear data structure is a structure wherein each data element can connect to more than two adjacent data elements. Examples of non linear 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 other one that links to the previous node.

47) What is Huffman's algorithm?

Huffman's algorithm is associated in creating extended binary trees that has minimum weighted path lengths from the given weights. It makes use of a table that contains 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 greatly 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.

Textbooks:

1. Ellis Horowitz and Sartaj Sahni, Fundamentals of Data Structures in C, 2nd Ed, Universities Press, 2014.
2. Seymour Lipschutz, Data Structures Schaum's Outlines, Revised 1st Ed, McGraw Hill, 2014.