MANIPAL INSTITUTE OF TECHNOLOGY

Manipal Academy of Higher Education

Manipal – 576 104

DEPARTMENT OF COMPUTER SCIENCE & ENGG.

CERTIFICATE

This is to certify that Ms./Mr. ……...…… Reg. No. …..…… Section: …… Roll No: ……... has satisfactorily completed the lab exercises prescribed for Data Structures Lab [CSE 2161] of Second Year B. Tech. Degree at MIT, Manipal, in the academic year 2021-22.

Date: …….....

Signature
Faculty in Charge

CONTENTS

Course Objectives

- To design and develop using structured approach
- To apply the recursive and iterative nature for solving various application problems
- To analyze the problem and solve it depending on static or dynamic nature of the inputs

Course Outcomes

At the end of this course, students will have the

- Apply recursion concepts to problems with arrays, functions, structures & pointers
- Implement applications using stacks and queues
- Solve problems using linked lists or trees

Evaluation plan

- Internal Assessment Marks : 60%

- End semester assessment of 2 hour duration: 40 %

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Be in time and adhere to the institution rules and maintain the decorum.
2. **Leave your mobile phones, pen drives and other electronic devices in your bag** and keep the bag in the designated place in the lab.
3. Must Sign in the log register provided
4. Make sure to occupy the allotted system and answer the attendance

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- Copy the program and results in the Lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- ***Academic honesty*** is required in all your work. You must solve all programming assignments entirely on your own, except where group work is explicitly authorized. This means you must not take, neither show, give or otherwise allow others to take your program code, problem solutions, or other work.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Programs should perform input validation (Data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
    - Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs and outputs.
    - Statements within the program should be properly indented.
    - Use meaningful names for variables and functions.
    - Make use of constants and type definitions wherever needed.
- The exercises for each week are divided under three sets:
    - Solved exercise
    - Lab exercises - to be completed during lab hours

    - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

THE STUDENTS SHOULD NOT

- **Possess mobile phones or any other electronic gadgets during the lab hours**.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

REVIEW OF C PROGRAMMING CONCEPTS: ARRAYS AND FUNCTIONS

Objectives:

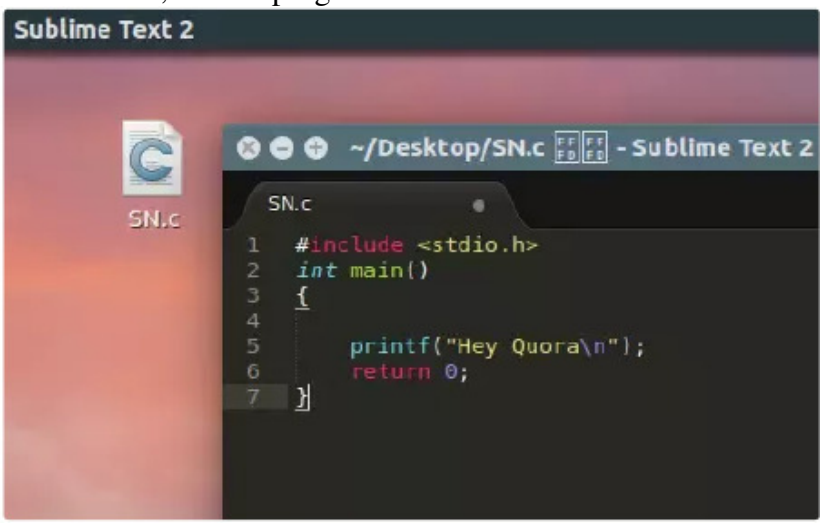
In this lab, student will be able to:

- Review C arrays and functions
- Familiarize with sublime editor for writing C programs
- Compile, execute and debug C program using ddd

I. SUBLIME EDITOR QUICK HELP GUIDE

Creating a source file:

1. Login to student account in Ubuntu
2. Press CNTRL + ALT+ T to open the *terminal*. Alternatively, choose *terminal* from dash home by typing the query 'terminal'
   EX: **user@user:~$** subl
3. Open sublime editor, write a program and save it with .c extension.



Building C programs using sublime:

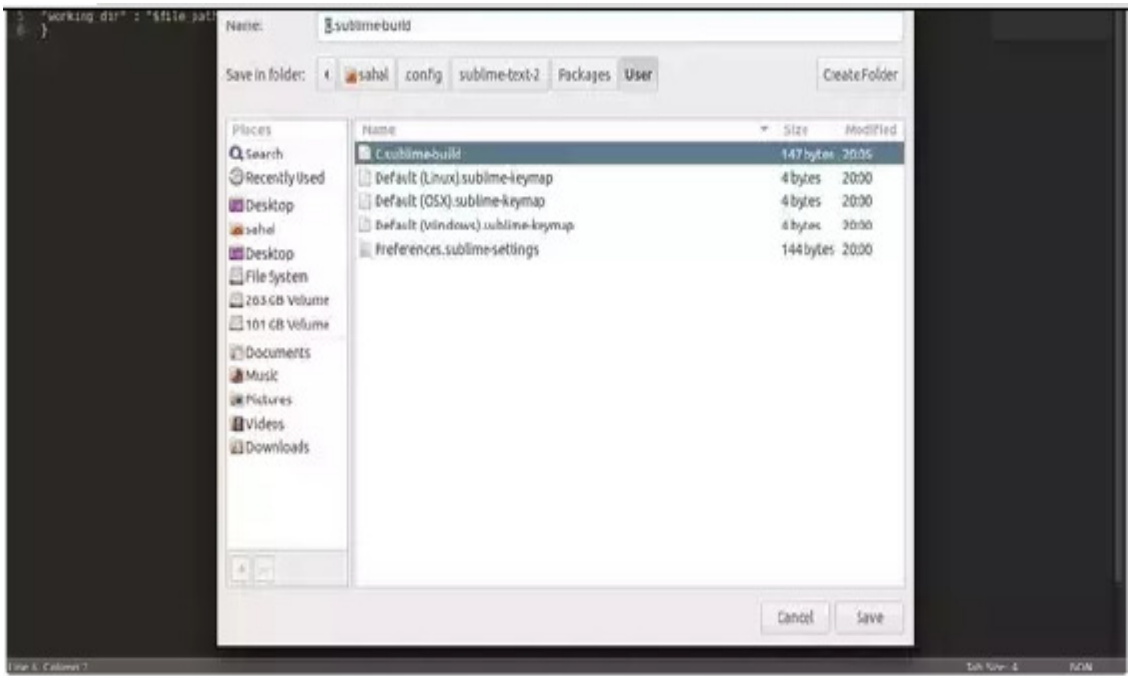
(i) Click **Tools -> Build System -> New Build System** and copy paste the following into curly braces:

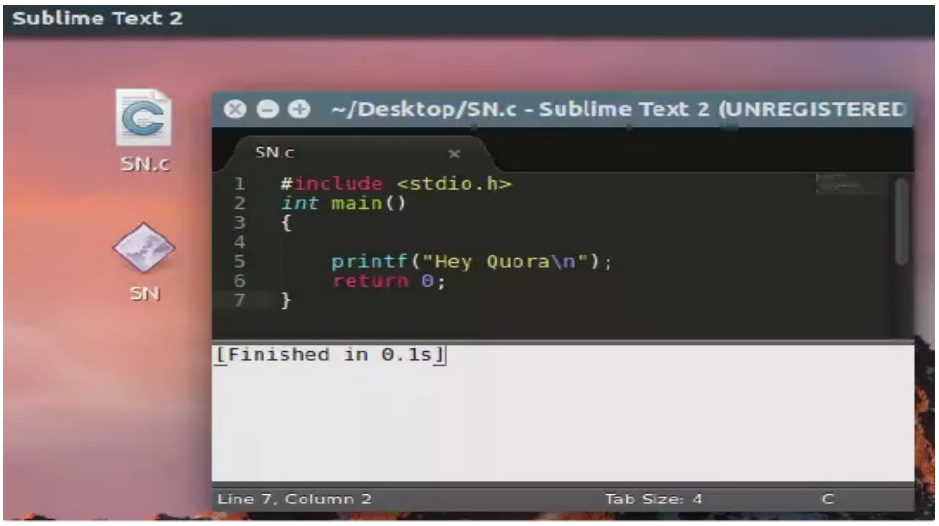
```
{
"cmd" : ["gcc  $file_name -o $file_base_name -lm -Wall"],
"selector" : "source.c",
"shell" : true,
 "working_dir" : "$file_path"
}
```

(ii) Select **File>Save** and name **c.sublime.build** without changing the directory.

(iii) Click **Tools>Build System** and select **c**.

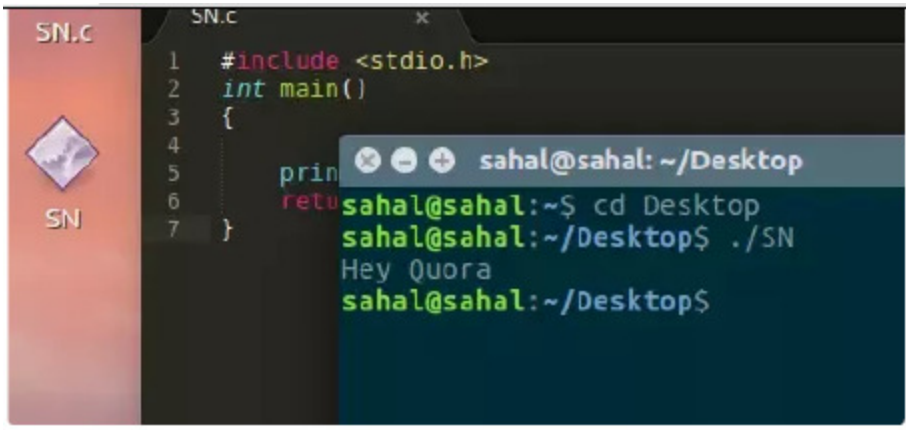
Now sublime is ready to build c programs [CNTRL+ B for compiling] in the editor.





Executing the Program:

1. Open terminal and locate your C program.
   Ex: **user@user:~$** cd Desktop
2. Run build file by typing:
   user@user:~$ ./SN



Compiling and Executing through terminal:

- Type the Compile command as: cc -g –o *outputfile inputcfile*
  - Ex:
    - cc –o *factorial factorial.c*

    In this case the executable file is created as *factorial.* To run the executable go to terminal and type ./factorial. (Here ./ refers to the

current directory, alternatively we can specify the absolute or relative path of the executable file).

- cc *factorial.c*

In this case the executable file is created as *a.out* by default. Remember this executable file will be over written when you run the same command again.

To run the executable, go to terminal and type ./a.out

Debugging through data display debugger:

- Open the terminal and run the ddd commands (refer ddd video)

II. SOLVED EXERCISE:

1) Write a C program to sort given list of n integers into ascending order using selection sort. Use function to sort.

Description: Assume we have an array 'A' with 'N' number of elements. This algorithm arranges elements in ascending order. 'Pass' is an index variable, which indicates the number of passes. The variable 'min_index' denotes the position of the smallest element encountered in that pass. 'I' is another index variable. The array and the size are passed to the function.

Algorithm: Selection Sort

Step 1: Using Pass index variable repeat the steps from first record to last – 1 records and perform all the steps 1to 4.

Step 2: Initialize the minimum index as
min_index = pass.

Step3: Obtain the element with the smallest value.

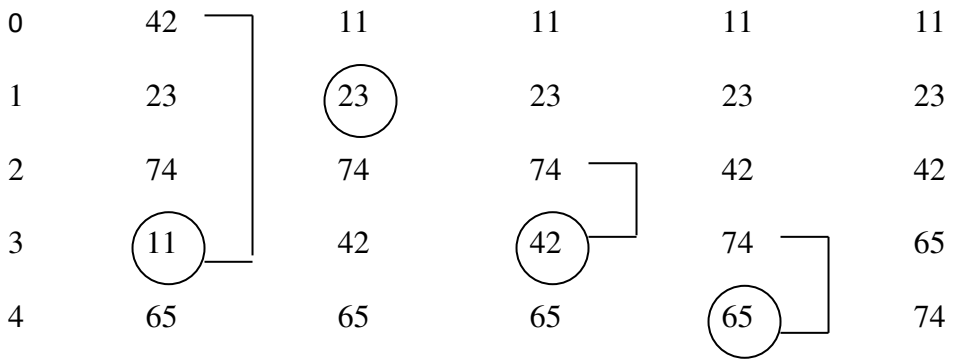
```
for(i= pass+1; i < N; i++)
 {
  if( A[i] < A [min_index])
            min_index = i;
 }
```

Step4: Exchange the elements

```
if(min_index != pass)
 {
      temp =  A[pass];
       A[pass] = A[min_index];
      A[min_index] = temp;
 }
```

Step5: Stop

Trace of Selection Sort:



File Name: selection_sort_fun.h

```
// function to swap array elements at positions x and y
void Swap(int arr[], int x, int y)
{
   int temp = arr[x];
   arr[x] = arr[y];
   arr[y] = temp;
}
/* function Selection Sort */
void SelectionSort(int arr[], int n)
{
   int pass, j, min_indx;

   // One by one move boundary of unsorted subarray
   for (pass = 0; pass < n-1; pass++)
   {
      // Find the minimum element in unsorted array
      min_indx = pass;
      for (j = pass+1; j < n; j++)
        if (arr[j] < arr[min_indx])
          min_indx = j;

      // Swap the found minimum element with the first element
      if (min_indx != pass)
        Swap(arr, min_indx, pass);
   }
}
```

File Name: selection_sort.c

```
#include <stdio.h>
#include "selection_sort_fun.h"
void main()
{
   int array[10];
   int i, j, n, temp;
   printf("Enter the value of n \n");
   scanf("%d", &n);
   printf("Enter the elements  \n");
   for (i = 0; i < n; i++)
```

```
    scanf("%d", &array[i]);
  /*  Selection sorting begins */
  SelectionSort(array, n);
  printf("The sorted list is (using selection sort): \n");
  for (i = 0; i < n; i++)
    printf("%d\t", array[i]);
  }
```

Sample Input and output:

Enter the total no of elements: 5
Enter the elements: 99 20 -12 43 34
The sorted list is (using selection sort): -12 20 34 43 99

III. LAB EXERCISE:

Write C programs to implement the following.

1) Find the sum of all the elements of an 1D double array of size n using a function **Add**. The values in the array are read from keyboard.
2) Implement an iterative Lsearch(….) function to search for an element in an 1D array of type integer using linear search technique.
3) Implement a C program to read, display and to find the product of two matrices using functions with suitable parameters. Check for the compatibility of the input matrices before multiplication.
4) Find the 2^{nd} largest in a list of numbers using a function (do not sort the list).

IV. ADDITIONAL EXERCISES:

1) **Random number generation and finding the frequency of occurrence**: Generate a large number of random numbers (say around 10K Samples). Each sample value should be between -100 to 100 (integers only). After generating the samples find the frequency of each distinct sample. Repeat the above steps with unknown size (hint: user will decide at run time). The purpose is to realize the advantages and disadvantages of using array.

2) **Addition of polynomials with two terms:** To perform different operations on polynomial with two terms x, y using 2-D array representations. Operations like addition and multiplication have to be implemented [ref: J.P Trembly]. If the 2D array representation is sparse then optimize the memory usage by using suitable alternative representation.

LAB NO: 2 **Date:**

SOLVING PROBLEMS USING ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION FUNCTIONS

Objectives:

In this lab, student will be able to:

i) Familiarize with syntax and usage of pointers and pointer to arrays and dynamic memory management functions

ii) Write C programs making use of pointer concepts and dynamic memory allocation functions

I. SOLVED EXERCISE:

1) Write a program to read n names of different sports and store them using array pointers. Use dynamic memory allocation and deallocation functions. The program should display all the names and deallocate the dynamic memory at the end of the program.

Description: The following figure illustrates the use of array of character pointers to store the names of different sports. First, an array of character pointers is created. Then, the name of different sports is read from the user and memory is allocated as per the input string length. This representation is memory efficient in comparison to the storage of multiple strings using 2D array of characters.

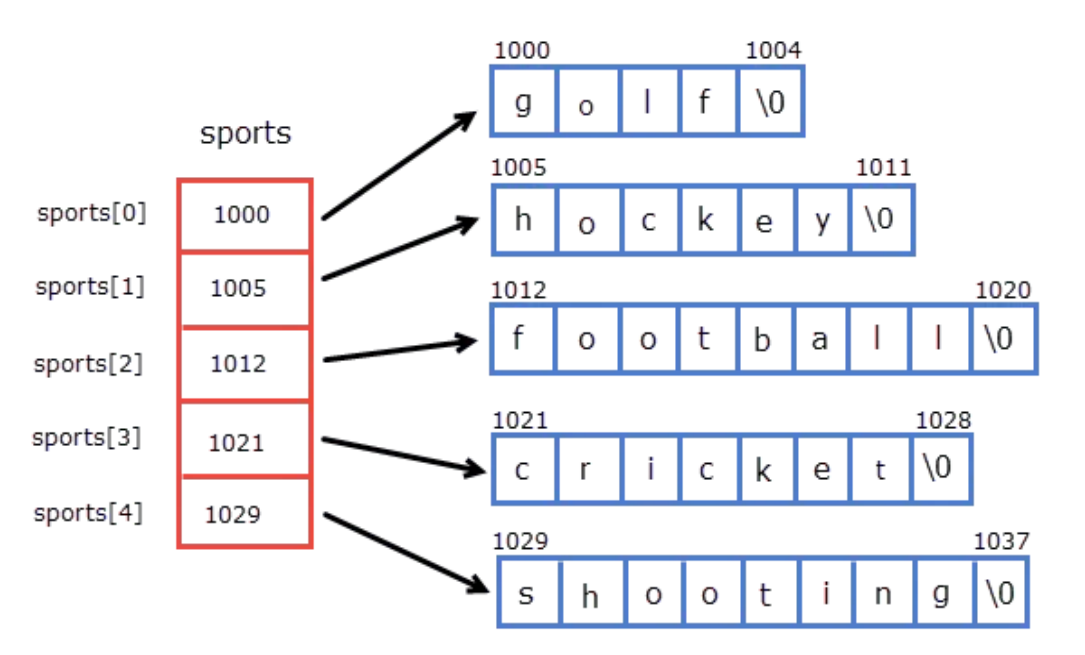


Fig. Memory representation of array of pointers

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(){
    int i,n;
    char *sports[10];
    char str[100];

    printf("\n enter the number of sports \n");
```

```
    scanf("%d", &n);
    printf("\nenter the names of sports:\n");
    for (i = 0; i < n; i++)
    {
        scanf("%s", str);
        //allocating memory equal to the length of string + 1
       //Last 1 byte to accommodate the '\0'
        sports[i] = (char*) calloc(strlen(str)+1, sizeof(char));
        strcpy(sports[i],str);
    }

    printf("\nGiven list of sports: \n");
    for (i = 0; i < n; i++)
            printf("%s\n", sports[i]);

   //Deallocate the dynamic memory
    for (i = 0; i < n; i++)
        free(sports[i]);

    return 0;
}
```

Sample input and output:

```
 enter the number of sports
5

enter the names of sports:
golf
hockey
football
cricket
shooting

Given list of sports:
golf
hockey
football
cricket
shooting

Process returned 0 (0x0)   execution time : 25.415 s
Press any key to continue.
```

II. LAB EXERCISES :

Note: Pass parameters using pointer to all the following functions.

1) Write a function Reverse to reverse the elements of an integer array using pointer. Access the elements of the array using dereference operator. Read the values from the keyboard in main function. Display the result in the main function.

2) Write a function Smallest to find the smallest element in an array using pointer. Create a dynamically allocated array and read the values from keyboard in main. Display the result in the main function.

3) Write a C program to
   a) Demonstrate passing pointers to a function.
   b) Demonstrate Returning pointer from a function.

c) Using pointer to pointer.

4) Implement a C program to read, display and to find the product of two matrices using functions with suitable parameters. Note that the matrices should be created using dynamic memory allocation functions and the elements are accessed using array dereferencing.

III. ADDITIONAL EXERCISES:

1) Write a C program to print an array in forward direction by adding one to pointer and in backward direction by subtracting one from pointer.
2) Write a function Palindrome to check whether a given string is a palindrome using pointers to array.

LAB NO: 3 **Date:**

SOLVING PROBLEMS USING RAGGED ARRAYS, STRUCTURES AND POINTERS

Objectives:

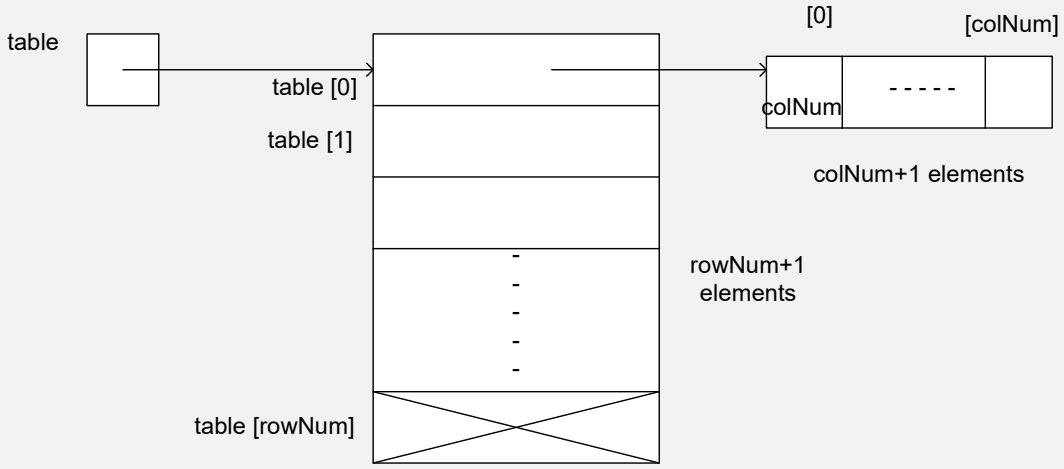
In this lab, student will be able to:

- Familiarize the usage of ragged arrays
- Write programs using structures and pointers
- Familiarize of dynamic memory allocation for structures

I. SOLVED EXERCISE:

1) Write a C program to implement a ragged array dynamically.

Description: In a ragged array the *table* pointer points to the first pointer in an array of pointers. Each array pointer points to a second array of integers, the first element of which is the number of elements in the list. A sample ragged array structure is shown below.



Algorithm: Construct a ragged array

Step 1: Declare a ragged array as a variable *table*.

Step 2: Ask the user for row size and set a variable – *rowNum*

Step 3: Allocate space for *(rowNum+1)* pointers as row pointers. The last row pointer will hold NULL

Step 4: Ask the user for column size and set a variable – *colNum*

Step 5: Allocate space for *(colNum+1)* data elements. The first element will hold value contained in colNum itself.

Step 6: Repeat step 3 for all rows

Step 7 : Display ragged array contents.

Step 8: Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
int rowNum, colNum, i, j;
int **table;
printf("\n enter the number of rows \n");
scanf("%d", &rowNum);
table = (int **) calloc(rowNum+1, sizeof(int *));
for (i = 0; i < rowNum; i++)  /* this will tell which row we are in */
{
        printf("enter size of %d row", i+1);
```

```
        scanf("%d", &colNum);
        table[i] = (int *) calloc(colNum+1, sizeof(int));
           printf("\n enter %d row elements ", i+1);
                   for (j = 1; j <= colNum;  j++)
                     {
                      scanf("%d", &table[i][j]);
                      }
        table[i][0] = colNum;
        printf("size of row number [%d] = %d", i+1, table[i][0]);
}
table[i] = NULL;
for (i = 0; i < rowNum; i++) /* this will tell which row we are in */
{
        printf("displaying %d row elements\n", i+1);
              for (j = 0; j <= *table[i];  j++)
                           {
                       printf("%5d", table[i][j]);
                           }
 printf("\n");
 }
 //freeup the memory
 for (i = 0; i < rowNum; i++) {
        free(table[i]);
    }
 free(table);
 return 0;
}
```

Sample input and output:

enter the number of rows: 3

enter size of row 1: 4

enter row 1 elements: 10 11 12 13

enter size of row 2: 5

enter row 2 elements: 20 21 22 23 24

enter size of row 3

enter row 3 elements: 30 31 32

displaying

10 11 12 13

20 21 22 23 24

30 31 32

II. LAB EXERCISES :

Note: Use Pointers to structures and dynamic memory management functions in the following programs.

1) Implement Complex numbers using structures. Write functions to add, multiply, subtract two complex numbers.

2) Write a C program to implement the following functions. Use pointers and dynamic memory management functions.

    i. To read one Student object where Student is a structure with name, roll number and CGPA as the data members

ii. To display one Student object

iii. To sort an array of Student structures according to the roll number.

3) Samuel wants to store the data of his employees, which includes the following fields: (i) Name of the employee (ii) Date of birth which is a collection of {day, month, year} (iii) Address which is a collection of {house number, zip code and state}. Write a 'C' program to read and display the data of N employees using pointers to array of structures.

4) Create a structure STUDENT consisting of variables of structures:

i. DOB {day, month (use pointer), year},

ii. STU_INFO {reg_no, name(use pointer), address},

iii. COLLEGE {college_name (use pointer), university_name}

where structure types from i to iii are declared outside the STUDENT independently. Show how to read and display member variables of DOB type if pointer variable is created for DOB inside STUDENT and STUDENT variable is also a pointer variable. The program should read and display the values of all members of STUDENT structure.

III. ADDITIONAL EXERCISES:

1) Design an application using suitable data structure to automate the process of class representative election. The application should take the inputs – No of students(voters) present in the class and No of Candidates and their name(s). During the voting process, the faculty on duty of conduction of election, need to initiate the voting (say by pressing a particular key during which a beep sound is heard). Then, a student can come and cast his/her vote against the candidate by entering the candidate's serial number. After casting a vote, a beep sound is given to confirm that the voting is done. This process is repeated until all students cast their vote. At the end, the total votes obtained by each candidate should be displayed along with number of foul votes. the candidate who has

won the maximum number of votes should be declared as class representative. (Hint: Use array of integers with array index as candidate number)

2) Design a suitable data structure for TEXT editor which includes operations such as inserting a line after the enter key is pressed and delete a line, search for a word in a line. When the key F2 is pressed, the content of the buffer should be saved in a file and displayed on the screen (Hint: Array of pointers to strings can be used as Data structure).

LAB NO: 4 **Date:**

SOLVING PROBLEMS USING RECURSION

Objectives:

In this lab, student will be able to:

- Formulate a recursive solution to a given problem
- Familiarize with recursion concept in C programs

I. SOLVED EXERCISE:

1) Write a C program to implement binary search

Title: IMPLEMENT BINARY SEARCH

1. Program to perform binary search on a set of keys.

Aim: To understand the working recursive function call and also binary search technique.

Description Binary search method employs the process of searching for a record only in half of the list, depending on the comparison between the element to be searched and the central element in the list. It requires the list to be sorted to perform such a comparison. It reduces the size of the portion to be searched by half after each iteration.

Algorithm: Binary Search

Assumption: The input array is in sorted order.

Step1: Given array A[low, high], find the value of mid location as mid =(low+high)/2

Step2: if Low > high return a Failure status and terminate the search; go to step 4.

Step3: Else Compare the key (element to be searched) with the mid element.

If key matches with middle element, we return the mid index; go to step 4.

Else If key is greater than the mid element, then key can only lie in right half sub-array after the mid element. So we recur for right half.

Else (x is smaller) recur for the left half until there are no more elements left in the array.

Step4: stop

Program:

File Name : binary_search_function.h

```
int bin_search(int low,int high,int item,int a[])
{
    int mid;
    if(low>high)
        return(-1);
    else
    {
        mid=(low+high)/2;
        if(item==a[mid])
            return(mid);
        else if(item<a[mid])
            return(bin_search(low,mid-1,item,a));
        else
            return(bin_search(mid+1,high,item,a));
    }
}
```

File Name : binary_search.c

```
#include <stdio.h>
#include "binary_search_function.h"
void main()
{
    int i, pos, a[30],n, item;
    printf("Enter number of items:");
    scanf("%d",&n);
    printf("Enter the elements in ascending order:\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
```

```
        printf("Enter element to be searched:");
        scanf("%d",&item);
        pos=bin_search(0,n-1,item,a);
        if(pos!=-1)
                printf("Item found at location %d",pos+1);
        else
                printf("Item not found");
}
```

Sample Input and Output:

Enter number of items: 6

Enter the elements in ascending order:

12 23 54 65 88 99

Enter element to be searched:99

Item found at location 6

II. LAB EXERCISES:

Write C programs using recursion. Also, write the call tree indicating call order.

1) To generate Fibonacci series with n terms.
2) To copy one string to another using Recursion.
3) To check whether a given String is Palindrome or not, using Recursion
4) Simulate the working of Tower of Hanoi for n disks. Print the number of moves.

III. ADDITIONAL EXERCISES:

1) To find the first capital letter in a string using Recursion.
2) To read a line of text and write it out backwards using Recursion

---

LAB NO:5 **Date:**

STACK CONCEPTS

Objectives:

In this lab, student will be able to:

- Understand stack as a data structure
- Design programs using stack concepts

I. SOLVED EXERCISE:

1) Write a c program to check if the given parenthesized expression has properly matching open and closing parenthesis.

Description: We use a stack to check the expression for matching opening and closing parenthesis. Here, the expression is scanned from start to end if an opening brace is encountered then it is pushed on to the stack. When a closing parenthesis is encountered a pop operation is performed. Ideally, if the number of opening and closing braces matches, then the stack will be empty after checking the entire expression.

Algorithm:

Step1: Set balanced to true

Step2: Set symbol to the first character in current expression

Step3: while (there are still more characters AND expression is still balanced)

    if (symbol is an opening symbol)

        Push symbol onto the stack

    else if (symbol is a closing symbol)

        if the stack is empty

```
                    Set balanced to false
                Else
                    Set openSymbol to the character at the top of the stack
                    Pop the stack
                    Set balanced to (symbol matches openSymbol)
            Set symbol to next character in current expression
Step4:  if (balanced)
                Write "Expression is well formed."
            else
                Write "Expression is not well formed."
   Step5: stop
```

Program:

File name: stack_operations.h

```
 # define MAX 10
 # define true 1
 # define false 0
 /* Structure definition */
 typedef struct
 {
        char item[MAX];
        int top;
 }stack;
 void push(stack *ps,char x);
 char pop(stack *ps);
 int empty(stack *ps);
 /* Push operation */
```

```
void push(stack *ps,char x)
{
        if (ps->top!=MAX-1)
        {
                ps->top++;
                ps->item[ps->top]=x;
        }
}

 /* Pop operation */
char pop(stack *ps)
{
        if(!empty(ps))
          return(ps->item[ps->top--]);
}

/* Stack empty operation */
int empty(stack *ps)
{
        if (ps->top==-1)
                return(true);
        else
                return(false);
}
```

File name: check_expr.c

```
#include <stdio.h>
#include <stdlib.h>
#include "stack_operations.h"
void main()
{
        char expn[25],c,d;
        int i=0;
        stack s;
        s.top=-1;
        printf("\n Enter the expression: ");
        gets(expn);
        while((c=expn[i++])!='\0')
        {
                if(c=='(')
                        push(&s,c);
                else
                        if(c==')')
                        {
                                d=pop(&s);
                                if(d!='(')
                                {
                                printf("\n Invalid Expression");
                                break;
                                }
                        }
        }
```

```
        if(empty(&s))
                printf("\n Balanced Expression");
        else
                printf("\n Not a Balanced  Expression");
 }
```

Sample Input and Output

Run 1:

Enter the expression: (a+b)+(c*d*(a-b)

Not a Balanced Expression

Run 2:

Enter the expression: (a+b)+(c*d*(a-b))

Balanced Expression

II. LAB EXERCISES:

Write a 'C' program to:

1) Implement a menu driven program to define a stack of characters. Include push, pop and display functions. Also include functions for checking error conditions such as underflow and overflow (ref. figure 1) by defining isEmpty and isFull functions. Use these function in push, pop and display functions appropriately. Use type defined structure to define a STACK containing a character array and an integer top. Do not use global variables.

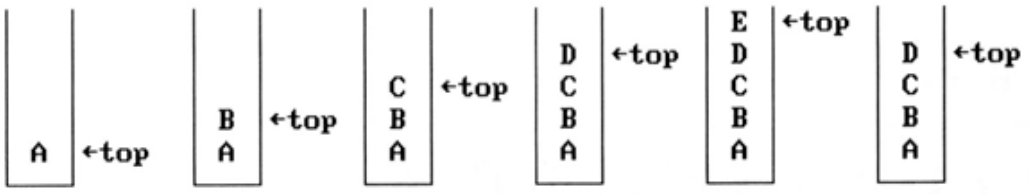


Figure 1: Inserting and deleting elements in a stack

2) Convert a given decimal number to binary using stack.

3) Determine whether a given string is palindrome or not using stack.

4) Given an array *arr* with n elements and a number k, k<n. The task is to delete k elements which are smaller than next element (i.e., we delete arr[i] if arr[i] < arr[i+1]) or become smaller than next because next element is deleted. Example:

   Input: arr[] = {20, 10, 25, 30, 40}, k = 2

   Output: 25 30 40

Explanation: First we delete 10 because it follows arr[i] < arr[i+1]. Then we delete 20 because 25 is moved next to it and it also starts following the condition.

III. ADDITIONAL EXERCISES:

1) Write a program to implement multiple stacks (say n stacks) in a single array. Implement ADD(i. X) and DELETE(i) to add X and delete an element from stack i, 1<= i <= n.

2) Given an array, print the Next Greater Element (NGE) for every element using stack. The Next Greater Element for an element x is the first greater element on the right side of x in array. Elements for which no greater element exist, consider next greater element as -1. For the input array [13, 7, 6, 12}, the next greater elements for each element are as follows.

Element		NGE
13	→	-1
7	→	12
6	→	12
12	→	-1

LAB NO: 6 **Date:**

STACK APPLICATIONS

Objectives:

In this lab, student will be able to:

- Identify the need for Stack data structure in a given problem.
- Develop c programs applying stack concepts

I. SOLVED EXERCISE:

1) Program for evaluation of postfix expression in C

Algorithm for Evaluation of Postfix Expression

Create an empty stack and start scanning the postfix expression from left to right.

- If the element is an operand, push it into the stack.
- If the element is an operator **O**, pop twice and get A and B respectively. Calculate B**O**A and push it back to the stack.
- When the expression is ended, the value in the stack is the final answer.

Evaluation of a postfix expression using a stack is explained in below example (fig. 2):

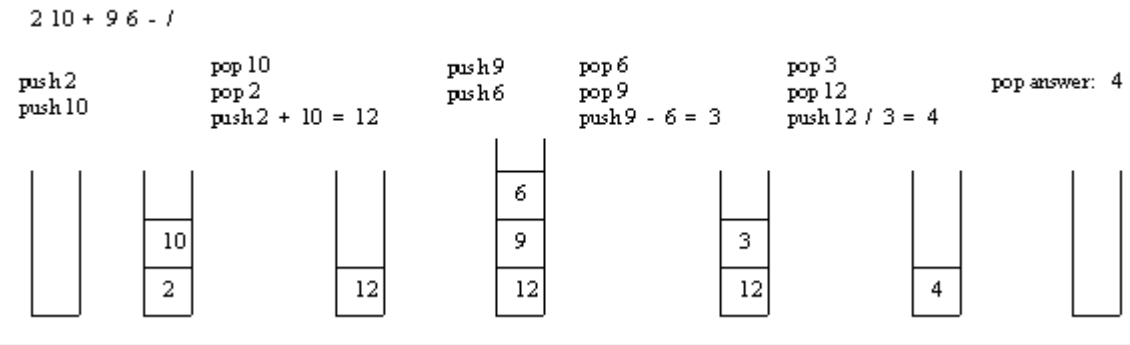


Figure 2: Illustrates the evaluation of a postfix expression using a stack

File name: eval_postfix_fun.h

```
#define MAX 20

typedef struct stack
{
   int data[MAX];
   int top;
}stack;

void init(stack *);
int empty(stack *);
int full(stack *);
int pop(stack *);
void push(stack *,int);
int evaluate(char x,int op1,int op2);

int evaluate(char x,int op1,int op2)
{
   if(x=='+')
      return(op1+op2);
   if(x=='-')
      return(op1-op2);
   if(x=='*')
      return(op1*op2);
   if(x=='/')
      return(op1/op2);
   if(x=='%')
      return(op1%op2);
}

void init(stack *s)
{
   s->top=-1;
}

int empty(stack *s)
{
   if(s->top==-1)
      return(1);
```

```
    return(0);
}

int full(stack *s)
{
    if(s->top==MAX-1)
        return(1);

    return(0);
}

void push(stack *s,int x)
{
    s->top=s->top+1;
    s->data[s->top]=x;
}

int pop(stack *s)
{
    int x;
    x=s->data[s->top];
    s->top=s->top-1;
    return(x);
}
```

File name:eval_postfix_expr.c

```
#include<stdio.h>
#include "eval_postfix_fun.h"
int main()
{
    stack s;
    char x;
    int op1,op2,val;
    init(&s);
    printf("Enter the expression(eg: 59+3*)\nsingle digit operand and operators
            only:");

    while((x=getchar())!='\n')
    {
```

```
            if(isdigit(x))
               push(&s,x-'0');       /*x-'0' for removing the effect of ascii  */
            else
            {
               op2=pop(&s);
               op1=pop(&s);
               val=evaluate(x,op1,op2);
               push(&s,val);
            }
         }
         val=pop(&s);
         printf("\nvalue of expression=%d",val);
         return 0;
      }
```

Sample Input and Output:
Enter the expression(eg: 59+3*)
single digit operand and operators only: 12+3*
value of expression= 9

II. LAB EXERCISES:

Write a C program to:

1) Evaluate a given prefix expression using stack.
2) Convert an infix expression to prefix.
3) Implement two stacks in an array.
4) To convert a prefix expression to postfix using stack.

III. ADDITIONAL EXERCISES:

1) Convert an infix expression to postfix.
2) Reverse a stack using recursion [Note: Only the IsEmpty, Push and Pop operations are allowed on stack]

LAB NO: 7 **Date:**

QUEUE CONCEPTS

Objectives:

In this lab, student will be able to:

- Understand queue as a data structure
- Design programs using queue concepts

I. SOLVED EXERCISE:

1) Implement a queue of integers. Include functions insertq, deleteq and displayq.

Description:

Whenever we perform an insertion the rear pointer is incremented by 1 and front remains the same. Whenever a deletion is performed the front is incremented by one. But in real life problem the front pointer will be in the same position the object which is in the queue shifts to the front. But when we implement a queue in computer the front moves, this is because if we shift the elements to the front the time complexity increases. To implement a linear queue we consider two pointers or markers front and rear. Initial value of front and rear is assumed to be -1

Algorithm: Queue Insert

Step 1. Check for overflow

If R== N-1

Then write ('OVERFLOW')

Return

Step2. else Increment rear pointer

R= R +1

Step3: Insert element

Q[R] = val

Step4: If F = -1

then F = 0

Return.

Algorithm: Queue Delete

Step1: Check for underflow

If F = = -1

Then Write ('UNDERFLOW')

Return

Step2: Delete an element

val = Q[F]

Step3: Queue empty?

if F > R

then F=R =-1

else F= F +1

Step4. Return an element

Return(val)

Step5: Stop

Program:

File name: queue_fun.h

```
#define MAX 20
typedef struct {
   int x[MAX];
   int front;
   int rear;
} queue;

void insertq(queue *, int);
void displayq(queue);
int deleteq(queue *);

void insertq(queue * q,int x)
 {
   if(q->rear==MAX)
   {
      printf("\nOverflow\n");
   }
   else
   {
      q->x[++q->rear]=x;
      if(q->front==-1)
      {
         q->front=0;
      }
   }
```

```
}

int deleteq(queue * q)
{
  int x;
  if(q->front==-1)
  {
     printf("\nUnderflow!!!\n");
  }
  else if(q->front==q->rear)
  {
     x=q->x[q->front];
     q->front=q->rear=-1;
     return x;
  }
  else
  {
     return q->x[q->front++];
  }
}

 void displayq(queue q)
{
  int i;
  if(q.front==-1&&q.rear==-1)
  {
    printf("\nQueue is Empty!!!");
```

```
    }
    else
    {
      printf("\nQueue is:\n");
      for(i=q.front;i<=q.rear;i++)
      {
        printf("%d\n",q.x[i]);
      }
    }
}
```

File name: queue.c

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

int main()
{
  queue q;
  q.front=-1;
  q.rear=-1;
  int ch,x,flag=1;
  while(flag)
    {
      printf("\n\n1. Insert Queue\n2. Delete Queue\n3. Display Queue\n4. Exit\n\n");
      printf("Enter your choice: ");
```

```
        scanf("%d",&ch);
        switch(ch)
        {
           case 1:
              printf("\nEnter the Element:");
              scanf("%d",&x);
              insertq(&q,x);
              break;
           case 2:
              x=deleteq(&q);
              printf("\nRemoved %d from the Queue\n",x);
              break;
           case 3:
              displayq(q);
              break;
           case 4:
              flag=0;
              break;
           default:
              printf("\nWrong choice!!! Try Again.\n");
        }
     }
  return 0;
}
```

II. LAB EXERCISES:

1) Implement a circular queue of Strings using structures. Include functions insertcq, deletecq and displaycq.

2) Implement two circular queues of integers in a single array where first queue will run from 0 to N/2 and second queue will run from N/2+1 to N-1 where N is the size of the array.

3) Write a function that takes a Queue and an element and returns true if the Queue contains this element or false if not. The elements in the Queue must remain in their original order once this method is complete. (use only queue operations)

4) Implement a queue with two stacks without transferring the elements of the second stack back to stack one. (use stack1 as an input stack and stack2 as an output stack)

III. ADDITIONAL EXERCISES:

1) Design a data representation sequentially mapping n queues into a single array A(1:m). Represent each queue as a circular queue within A. Write algorithms ADDQ, DELETEQ and QUEUE-FULL for this representation. Illustrate the working with a menu driven program.

2) Design a data representation, sequentially mapping n data objects into an array A(1:m). n1 of these data objects are stacks and the remaining n2 = n - n1 are queues. Write algorithms to add and delete elements from these data structures. Use the same SPACE_FULL algorithm for both types of data structures. This algorithm should provide space for the i-th data object if there is some space not currently being used. Note that a circular queue with space for r elements can hold only r – 1 elements. Illustrate the working by writing a c program.

LAB NO: 8 **Date:**

QUEUE APPLICATIONS

Objectives:

In this lab, student will be able to:

- Identify the need for queue data structure in a given problem.
- Develop c programs applying queue concepts

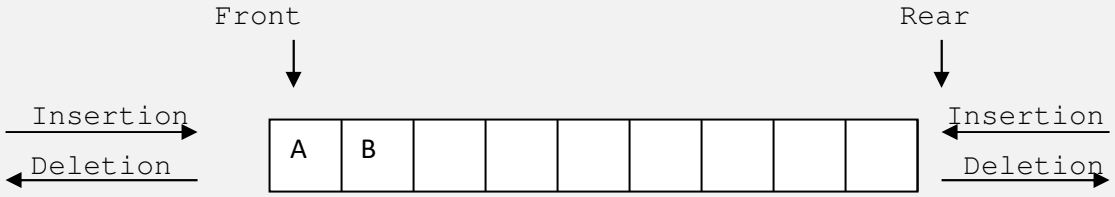
I. SOLVED EXERCISE:

1) Implement a dequeue of integers with following functions.

a) deleteLeft b) addLeft c) deleteRight d) addRight e) display

Description:

A queue that supports insertion and deletion at both the front and rear is called **double-ended queue or Dequeue.** A Dequeue is a linear list in which elements can be added or removed at either end but not in the middle.



The operations that can be performed on Dequeue are

1. Insert to the beginning
2. Insert at the end
3. Delete from the beginning
4. Delete from end

There are two variation of Dequeue namely, an input-restricted Dequeue and an Output restricted Dequeue – which are intermediate between a Dequeue and a queue. Specifically, an input-restricted Dequeue is a Dequeue which allows insertions at only

one end of the list but allows deletions at both ends of the list; and an output-restricted Dequeue is a Dequeue which allows deletions at only one end of the list allows insertion at both ends of the list.

Algorithm: Insert End (rear)

Step1: if(front==MAX/2)

Write("Queue Full: Can't enter more elements at the end of queue");

return;

Step2: Else Insert the element and update the front pointer i.e queue[front]=token;

front=front+1;

Algorithm: Insert Start (front)

Step1: If (front==MAX/2)

Write("Queue Full: can't enter more elements at the start of queue");

return;

Step2: Insert the element by updating the rear pointer i.e

rear=rear-1;

queue[rear]=token;

Algorithm: Delete End (rear)

Step1: If(front==rear)

Write("Queue empty");

return 0;

Step2: Otherwise update the front and return the element i.e

```
            front=front-1;
            t=queue[front+1];
            return t;
```

Algorithm: Delete Start (front)

Step1: If(front==rear)

```
            Write("\nQueue empty");
            return 0;
```

Step2: Update the rear pointer and return the element i.e

```
            rear=rear+1;
            t=queue[rear-1];
            return t;
```

Program:

File name: deque_fun.h

```
#define MAX 30
typedef struct dequeue
{
   int data[MAX];
   int rear,front;
}dequeue;

void initialize(dequeue *p);
int empty(dequeue *p);
int full(dequeue *p);
```

```
void enqueueR(dequeue *p,int x);
void enqueueF(dequeue *p,int x);
int dequeueF(dequeue *p);
int dequeueR(dequeue *p);
void print(dequeue *p);

void initialize(dequeue *P)
{
   P->rear=-1;
   P->front=-1;
}

int empty(dequeue *P)
{
   if(P->rear==-1)
      return(1);
   return(0);
}

int full(dequeue *P)
{
   if((P->rear+1)%MAX==P->front)
      return(1);

   return(0);
}
void enqueueR(dequeue *P,int x)
```

```
{
   if(empty(P))
   {
      P->rear=0;
      P->front=0;
      P->data[0]=x;
   }
   else
   {
      P->rear=(P->rear+1)%MAX;
      P->data[P->rear]=x;
   }
}

void enqueueF(dequeue *P,int x)
{
   if(empty(P))
   {
      P->rear=0;
      P->front=0;
      P->data[0]=x;
   }
   else
   {
      P->front=(P->front-1+MAX)%MAX;
      P->data[P->front]=x;
   }
```

```
}

int dequeueF(dequeue *P)
{
   int x;
   x=P->data[P->front];
   if(P->rear==P->front)    /*delete the last element */
      initialize(P);
   else
      P->front=(P->front+1)%MAX;
   return(x);
}

int dequeueR(dequeue *P)
{
   int x;
   x=P->data[P->rear];
   if(P->rear==P->front)
      initialize(P);
   else
      P->rear=(P->rear-1+MAX)%MAX;
   return(x);
}

void print(dequeue *P)
{
   if(empty(P))
```

```
    {
        printf("\nQueue is empty!!");
        exit(0);
    }
    int i;
    i=P->front;
    while(i!=P->rear)
    {
        printf("\n%d",P->data[i]);
        i=(i+1)%MAX;
    }
    printf("\n%d\n",P->data[P->rear]);
}
```

File name: dequeuer.c

```
#include<stdio.h>
#include<process.h>
#include "dequeue_fun.h"
int main()
{
    int i,x,op,n;
    dequeue q;
    initialize(&q);
    do
    {
```

```
printf("\n1.Create\n2.Insert(rear)\n3.Insert(front)\n4.Delete(rear)\n5.Delete(front)");
      printf("\n6.Print\n7.Exit\n\nEnter your choice:");
      scanf("%d",&op);
      switch(op)
      {
         case 1: printf("\nEnter number of elements:");
               scanf("%d",&n);
               initialize(&q);
               printf("\nEnter the data:");

               for(i=0;i<n;i++)
               {
                  scanf("%d",&x);
                  if(full(&q))
                  {
                     printf("\nQueue is full!!");
                     exit(0);
                  }
                  enqueueR(&q,x);
               }
               break;

         case 2: printf("\nEnter element to be inserted:");
               scanf("%d",&x);

               if(full(&q))
```

```
            {
               printf("\nQueue is full!!");
               exit(0);
            }
            enqueueR(&q,x);
            break;

      case 3: printf("\nEnter the element to be inserted:");
            scanf("%d",&x);
            if(full(&q))
            {
               printf("\nQueue is full!!");
               exit(0);
            }
            enqueueF(&q,x);
            break;

      case 4: if(empty(&q))
            {
               printf("\nQueue is empty!!");
               exit(0);
            }
            x=dequeueR(&q);
            printf("\nElement deleted is %d\n",x);
            break;

      case 5: if(empty(&q))
```

```
            {
               printf("\nQueue is empty!!");
               exit(0);
            }
            x=dequeueF(&q);
            printf("\nElement deleted is %d\n",x);
            break;

        case 6: print(&q);
            break;
        default: break;
      }
   }while(op!=7);
   return 0;
}
```

II. LAB EXERCISES:

1) Implement an ascending priority queue.

   Note: An ascending priority queue is a collection of items into which items can be inserted arbitrarily and from which only the smallest item can be removed. If apq is an ascending priority queue, the operation pqinsert(apq,x) inserts element x into apq and pqmindelete(apq) removes the minimum element from apq and returns its value.

2) Implement a queue of strings using an output restricted dequeue (no deleteRight).
   Note: An output-restricted deque is one where insertion can be made at both ends, but deletion can be made from one end only, where as An input-restricted deque is

one where deletion can be made from both ends, but insertion can be made at one end only.

3) Write a program to check whether given string is a palindrome using a dequeue.

4) Given a queue of integers, write a program to reverse the queue, using only the following operations:

   i. enqueue(x): Add an item x to rear of queue.
   ii. dequeue() : Remove an item from front of queue.
   iii. empty() : Checks if a queue is empty or not.

III. ADDITIONAL EXERCISES:

1) Implement a queue of integers using an input restricted dequeue (no addRight).

2) Write a program to simulate airport traffic control for a small but busy airport assuming the following specifications.

   - There is only one runway for both landing and takeoff.
   - In each unit time, only one plane can land or take off but not both.
   - Planes arrive and take off at random time, so that at any given unit, the runway may be idle or a plane may be landing or taking off.
   - Number of planes waiting to take off or land should not exceed a certain fixed limit.
   - Landing planes should have high priority than Taking-off plane.
   - After running for some period, the program should print statistics such as: number of planes processed, landed, and took off, refused; average landing and take-off waiting times, and average run-way idle time.

LAB NO: 9 **Date:**

LINKED LIST CONCEPTS

Objectives:

In this lab, student will be able to:

- Understand list as a data structure
- Design programs using linked list concepts

I. SOLVED EXERCISE:

1) Implement stack using singly linked list.

Description: Implementing a stack using linked list is performed by calling insert beginning for Push operation and calling delete beginning for a pop operation. Push is performed by adding a node to the beginning of the list and updating the header. Pop operation is defined as deleting a node from the beginning of the list and updating the header accordingly.

Algorithm: Push

Step1: Create a new node say

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

Step2: Check whether a node has been created or not if newnode is

NULL node is not created

i.e. if(newnode= =NULL)

{ printf("Out of memory");

exit(0);

}

else

Step3: Insert the data in data field

newnode->data = element;

Step4: Check whether the list is empty, if so then hold the address of newnode **top**.

i.e if(top= =NULL)

{ top =newnode;

top->link=NULL;

}

else

Step5: add the new node to the top end

newnode->link=top;

And update the top i.e top=newnode;

Step6: Stop.

Algorithm: POP

Step1: Make temp to point to the top element

temp=top;

Step2: make the node next to the list pointer as the beginning of list

top = top->link; //equivalent to top=top-1 in array implementation

Step3: separate temp from chain

temp->link=NULL

delete the first node pointed by temp

free (temp);

Step4: stop

Program:

File name: stack_sll_fun.h

```
typedef struct node
{       int info;
        struct node *link;
}NODE;

NODE* push(NODE *list,int x)
{       NODE *new,*temp;
        new=(NODE*) malloc(sizeof(NODE));
        new->link=list;
        new->info=x;
        return(new);
}

NODE* pop(NODE *list)
{       NODE *prev,*temp;
        if(list==NULL)
        {
                printf("\nStack Underflow\n");
                return(list);
        }
        temp=list;
        printf("Deleted element is %d",temp→info);
        free(temp);
        list = list→link;
        return(list);
```

```
}

void display(NODE *list)
{
        NODE *temp;
        printf("\n\nSTACK:");
        if(list==NULL)
        {
                printf("  Stack is empty");

        printf("\n\n**********************************************");
                return;
        }
        temp=list;
        while(temp!=NULL)
        {
                printf("%5d",temp->info);
                temp=temp->link;
        }
        printf(" <- TOP");
        printf("\n\n**********************************************");
}

int getchoice()
{
        int ch;
        printf("**********************************************\n\n");
```

```
        printf("--------------------Menu--------------------\n");
        printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ch);
        return(ch);
}
```

File name: stack_sll.c

```
#include<stdio.h>
#include "stack_sll_fun.h"
int main()
{
        NODE *list;
        int x,ch;
        list=NULL;
        while(1)
        {
                ch=getchoice();
                switch(ch)
                {
                        case 1: printf("Enter the element to be pushed:");
                                scanf("%d",&x);
                                list=push(list,x);
                                display(list);
                                break;
                        case 2: list=pop(list);
                                display(list);
```

```
                                        break;
                                case 3: display(list);
                                        getch();
                                        break;
                                case 4: exit(1);
                                default: printf("\nInvalid choice");
                printf("\n\n*********************************************");
                        }
                }
                return 0;
}
```

Sample Input and Output

```
**********************************************
--------------------Menu--------------------
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:1
Enter the element to be pushed:23

STACK:   23 <- TOP

**********************************************
--------------------Menu--------------------
1. Push
```

```
2. Pop
3. Display
4. Exit
Enter your choice:
1
Enter the element to be pushed:34

STACK:   23   34 <- TOP

************************************************
----------Menu----------
1. Push
2. Pop
3. Display
4. Exit
Enter your choice:
2
Popped element is 34
STACK:   23 <- TOP
************************************************
```

II. LAB EXERCISES:

1) Implement a queue using singly linked list without header node.
2) Perform UNION and INTERSECTION set operations on singly linked lists with and without header node.

3) Create a circular singly linked LIST by merging two sorted singly circular linked lists containing char data, such that the final LIST is sorted

4) You're given the pointer to the head node of a sorted singly linked list, where the data in the nodes is in ascending order. Delete as few nodes as possible so that the list does not contain any value more than once (deleting duplicates). The given head pointer may be null indicating that the list is empty.

III. ADDITIONAL EXERCISES:

1) Write a program to implement the Josephus Circle Problem: There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of persons, n and a number m which indicates that m-1 persons are skipped and m^{th} person is killed in circle. The task is to choose the place in the initial circle so that you are the last one remaining and so survive.

2) Reverse a singly linked list containing words in the data field using recursion.

LAB NO: 10 **Date:**

LINKED LIST APPLICATIONS

Objectives:

In this lab, student will be able to:

- Identify the need for list data structure in a given problem.
- Develop c programs applying linked concepts

I. SOLVED EXERCISE:

1) Given two polynomials, write a program to perform the addition of two polynomials represented using doubly circular linked list with header and display the result.

Description: Suppose we wish to manipulate polynomials of the form $p(x) = c_1*x\text{^}e_1 + c_2*x\text{^}e_2 + \ldots + c_n* x\text{^} e_n$, where $e_1 > e_2 > \ldots > e_n \geq 0$. Such a polynomial can be represented by a linked list in which each cell has three fields: one for the coefficient c_i, one for the exponent e_i, and one for the pointer to the next cell. For example, the polynomial $4x^2+2x+4$, can be viewed as list of the following pairs (4,2),(2,1),(4,0). Therefore, we can use a linked list in which each node will have four fields to store coefficient, exponent and two link fields. The right link will be pointing to the next node and left link will be pointing to the previous node. The last node's right link will point to the header in a circular list and the left link of header is made to point to the last node. A dummy node is maintained as head to the list.

Algorithm: Add polynomials

Step1: Take references to the header of first and second polynomial list

one=h1->rlink; two=h2->rlink; and h3 is a pointer to the resulting list.

Step2: Traverse through the lists by checking the exponents until either of the pointer is not null, i.e, While(one!=h1 && two!=h2) do the following

Step3: If the exponents of botht the lists are equals, add the coefficients and insert added coefficient and the exponent to a resultant list i.e

```
if((one->ex)==(two->ex))
    {       h3=add(h3,((one->info)+(two->info)),one->ex);
            one=one->rlink;
            two=two->rlink;
    }
```

Step4: Else if exponent of first list pointer is greater, then insert first list pointer exponent and coefficient to result list. Update the first lists pointer only to the next node and continue with step2.

```
            h3=add(h3,one->info,one->ex);
        one=one->rlink;
```

Step5: Else insert second list pointer exponent and coefficient to result list. Update the second lists pointer only to the next node and continue with step2.

```
        h3=add(h3,two->info,two->ex);
        two=two->rlink;
```

Step6: If there are any terms left in second list copy it to the resultant list i.e

```
while(two!=h2)
    {       h3=add(h3,two->info,two->ex);
            two=two->rlink;
    }
```

Step7: If there are any terms left in first list copy it to the resultant list i.e

```
while(one!=h1)
    {       h3=add(h3,one->info,one->ex);
```

```
                one=one->rlink;
        }
 Step8:         return a pointer to the resultant list h3
```

Program:

File name: poly_add_dll_fun.h

```
struct node
{       int info;
        int ex;
        struct node *llink;
        struct node *rlink;
};
typedef struct node *NODE;
NODE add(NODE head,int n,int e)
{
       NODE temp,last;
        temp=(NODE)malloc(sizeof(struct node));
        temp->info=n;
        temp->ex=e;
        last=head->llink;
        temp->llink=last;
        last->rlink=temp;
        temp->rlink=head;
```

```
        head->llink=temp;
        return head;
}
NODE sum(NODE h1,NODE h2,NODE h3)
{
        NODE one,two;
        one=h1->rlink;
        two=h2->rlink;
        while(one!=h1 && two!=h2)
        {       if((one->ex)==(two->ex))
                {       h3=add(h3,((one->info)+(two->info)),one->ex);
                        one=one->rlink;
                        two=two->rlink;
                }
                else if(one->ex>two->ex)
                {       h3=add(h3,one->info,one->ex);
                        one=one->rlink;
                }
                else
                {       h3=add(h3,two->info,two->ex);
                        two=two->rlink;
                }
        }
        while(two!=h2)
                {       h3=add(h3,two->info,two->ex);
                        two=two->rlink;
                }
```

```
        while(one!=h1)
                {       h3=add(h3,one->info,one->ex);
                        one=one->rlink;
                }
        return h3;
}

void display(NODE head)
{       printf("\ncontents of list are\n");
        NODE temp=NULL;
        temp=head->rlink;
        while(temp!=head)
        {       printf("%d %d\t",temp->info,temp->ex);
                temp=temp->rlink;
        }
}
```

File name: ploy_add_dll.c

```
#include<stdio.h>
#include<stdlib.h>
#include “poly_add_dll_fun.h”
int main()
{       int m,n,e,k;
```

```
NODE h1,h2,h3,h4;
h1=(NODE)malloc(sizeof(struct node));
h2=(NODE)malloc(sizeof(struct node));
h3=(NODE)malloc(sizeof(struct node));
h4=(NODE)malloc(sizeof(struct node));
h1->rlink=h1;
h1->llink=h1;
h2->rlink=h2;
h2->llink=h2;
h3->rlink=h3;
h3->llink=h3;
h4->rlink=h4;
h4->llink=h4;
printf("\nnumber of nodes in list1\n");
scanf("%d",&n);
while(n>0)
{       scanf("%d",&m);
        scanf("%d",&e);
        h1=add(h1,m,e);
        n--;
}
display(h1);
printf("\nnumber of nodes in list2\n");
scanf("%d",&k);
while(k>0)
{       scanf("%d",&m);
        scanf("%d",&e);
```

```
                h2=add(h2,m,e);
                k--;
        }
        display(h2);
        printf("\nthe sum is\n");
        h3=sum(h1,h2,h3);
        display(h3);
        return 1;
}
```

Sample Input and Output

```
number of nodes in list1
3
3 3 3 2 4 1
contents of list are
3 3     3 2     4 1
number of nodes in list2
3
2 3 2 2 1 1
contents of list are
2 3     2 2     1 1
the sum is
contents of list are
5 3     5 2     5 1
```

II. LAB EXERCISES:

1) Write a menu driven program to implement doubly linked list without header node to insert into and delete from both the sides.
2) Add two long positive integers represented using circular doubly linked list with header node.
3) Reverse a doubly linked list containing words in the data field.
4) Given two polynomials, write a program to perform the following operations on singly circular linked list with header node. Use menu driven approach to input two polynomials, subtract, multiply and display the result.

III. ADDITIONAL EXERCISES:

1) Implement Union and Intersection operation using doubly linked list.
2) Write a 'C' program to represent a sparse matrix using doubly linked lists. A Matrix with most of its elements being zero is called a sparse matrix. Example

$$\begin{pmatrix} 0 & 3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 8 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 4 & 0 \\ 0 & 1 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 7 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \end{pmatrix}$$

LAB NO: 11 **Date:**

TREE CONCEPTS

Objectives:

In this lab, student will be able to:

- Understand tree as a data structure
- Design programs using tree concepts

I. SOLVED EXERCISE:

1) Create a binary tree using recursion and display its elements using all the traversal methods.

Description: To create and maintain the information stored in a binary tree, we need an operation that inserts new nodes into the tree. We use the following insertion approach. A new node is made root for the first time and there after a new node is inserted either to the left or right of the node. -1 one is entered to terminate the insertion process. This is recursively done for left subtree and the right subtree respectively.

Program:

File name: binary_tree_recursion_fun_1.h

```
typedef struct node
{
   int data;
   struct node *lchild;
   struct node *rchild;     } *NODE;

NODE Create_Binary_Tree()
```

```
{
   NODE temp;
   int ele;
   printf("Enter the element to inserted (-1 for no data):");
   scanf("%d",&ele);
   if(ele==-1)
      return NULL;
   temp=(NODE*)malloc(sizeof(struct node));
   temp->data=ele;
   printf("Enter lchild child of %d:\n",ele);
   temp->lchild=create();

   printf("Enter rchild child of %d:\n",ele);
   temp->rchild=create();
   return temp;
}
void inorder(NODE *ptr)
{
        if(ptr!=NULL)
        {
                inorder(ptr->lchild);
                printf("%5d",ptr->info);
                inorder(ptr->rchild);
        }
}

void postorder(NODE *ptr)
```

```
        {
                if(ptr!=NULL)
                {
                        postorder(ptr->lchild);
                        postorder(ptr->rchild);
                        printf("%5d",ptr->info);
                }
        }

        void preorder(NODE *ptr)
        {
                if(ptr!=NULL)
                {
                        printf("%5d",ptr->info);
                        preorder(ptr->lchild);
                        preorder(ptr->rchild);
                }
}
```

File name: binary_tree.c

```
        #include<stdio.h>
        #include "binary_tree_recursion_fun_1.h"
        int main()
        {
                int n,x,ch,i;
                NODE *root;
                root=NULL;
                while(1)
```

```
        {
        printf("********************Output********************\n\n");
                printf("-----------Menu-----------\n");
                printf(" 1. Insert\n 2. All traversals\n 3. Exit\n");
                printf("Enter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("Enter node :\n");
                                scanf("%d",&x);
                                root=Create_Binary_Tree();
                                break;
                        case 2: printf("\nInorder traversal:\n");
                                inorder(root);
                                printf("\nPreorder traversal:\n");
                                preorder(root);
                                printf("\nPostorder traversal:\n");
                                postorder(root);

        printf("\n\n*********************************************");
                                break;
                        case 3: exit(0);
                }
        }
        return 0;
}
```

Sample Input and Output

```
**********Output**********

----------Menu----------
 1. Insert
 2. All traversals
 3. Exit
Enter your choice:1
Enter node:
20      25      -1      30      40      -1      -1      -1
**********Output**********

Enter your choice:2

Inorder traversal:      25      30      20      40      28
Preorder traversal:     20      25      30      28      40
Postorder traversal:    30      25      40      28      20

          ******************************************
```

II. LAB EXERCISES:

1) Implement Binary Tree(BT) using iterative function and display the elements of the BT using recursive in-order, pre-order, post-order traversal methods.

2) Display the elements of Binary Tree created using iterative preorder, post-order (Use single stack), in-order and level-order traversals.

3) Create an expression tree for the given postfix expression and evaluate it.

4) Implement recursive functions to do the following:

   a) To create a copy of a BT.

   b) Testing for equality of two BTs.

III. ADDITIONAL EXERCISES:

1) Implement trees using arrays.

2) Create an expression tree for the given prefix expression and evaluate it.

LAB NO: 12 **Date:**

TREE APPLICATIONS

Objectives:

In this lab, student will be able to:

- Identify the need for tree data structure in a given problem.
- Develop c programs applying tree concepts

I. SOLVED EXERCISE:

1) Write program to find the depth, height, number of leaves, nodes in a BST

Description: We can determine the number of nodes in the tree if we know the number of nodes in the left subtree and the number of nodes in the right subtree. That is, the number of nodes in a tree is

Number of nodes in left subtree + number of nodes in right subtree + 1

This is easy. Given a function CountNodes and a pointer to a tree node, we know how to calculate the number of nodes in a subtree; we call CountNodes recursively with the pointer to the subtree as the argument. Similarly we can find the height by taking the maximum of the depth of recursion on left subtree and right subtree. Here every time each recursive call increments the depth of recursion by 1. T find the leaf nodes we need to check the nodes which do not have any children. This can also be don recursively by exploring left and right subtree if it exists.

Algorithm: Count Nodes

Step1: if tree is NULL

return 0

Step2: else

return CountNodes(Left(tree)) + CountNodes(Right(tree)) + 1

Algorithm: height

Step1: if tree is NULL

return 0

Step2: else

return 1+ max(height(Left(tree)), height(Right(tree)));

(where max is the maximum of two values)

Algorithm: CountLeaf

Step1: if tree==NULL return;

Step2: CountLeaf(Left(tree));

Step3: if(Left(tree)==NULL && Right(tree)==NULL) leaf++;

Step4: countleaf(Right(tree));

Program:

File name: binary_search_tree_recursion_fun_2.h

```
int c_node = 0, leaf = 0;

int max(int a, int b)
{
        return (a>b)?a:b;
}
```

```
int height(NODE *ptr)
{
        if(ptr==NULL) return -1;
        return 1+max(height(ptr->lchild),height(ptr->rchild));
}

void count_node(NODE *ptr)
{
        if(ptr==NULL) return;
        count_node(ptr->lchild);
        c_node++;
        count_node(ptr->rchild);
}

void count_leaf(NODE *ptr)
{
        if(ptr==NULL) return;
        count_leaf(ptr->lchild);
        if(ptr->lchild==NULL && ptr->rchild==NULL) leaf++;
        count_leaf(ptr->rchild);
}
```

File Name: binary_search_tree_recursion_2.c

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

```
#include "binary_search_tree_recursion_fun_1.h"
#include "binary_search_tree_recursion_fun_2.h"

int main()
{
        int h,x,ch,i;
        NODE *root;
        root=NULL;
        while(1)
        {

        printf("********************Output********************\n\n");
                printf("-----------Menu-----------\n");
                printf(" 1. Create\n 2. Height,Count the nodes and leaves\n 3.
Exit\n");
                printf("Enter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                           printf("Enter node (do not enter duplicate nodes):\n");
                                scanf("%d",&x);
                                root=create(root,x);
                                break;
                        case 2: printf("\nTree Generated!\nPrinting Now in
                                indorder - ");
                                inorder(root);
```

```
                                        int h = height(root);
                                        count_node(root);
                                        count_leaf(root);
                                        printf("\nHeight - %d",h);
                                        printf("\nNo of Nodes - %d",c_node);
                                         printf("\nNo of Leaves - %d",leaf);
                                         printf("\n\n**************************");
                                        break;
                                case 3: exit(0);
                        }
                }
                return 0;
}
```

II. LAB EXERCISES:

1) Write a menu driven program to do the following using iterative functions:

   i. To create a BST for a given set of integer numbers

   ii. To delete a given element from BST .

   Note: Replace the element to be deleted with the largest element in the LST (Left Sub Tree)) when the node to be deleted has both LST (Left Sub Tree) and RST (Right Sub Tree).

   iii. To display the elements using in-order traversal.

2) Given a BST, write a function to find the in - order successor of a node.

3) Given a Binary Tree and a key, write a function that prints all the ancestors of the key in the given binary tree.

4) Write a program to construct a BT for given pre-order and the post- order traversal sequences.

III. ADDITIONAL EXERCISES:

1) Phone book application for student records using roll number as key: Implement a menu driven program to create a phone book for student records where in each student record will contain roll-no, name, section, CGPA and phone number. The phone book must have an option to search for a record based on roll no of the student. Search is done using a Binary Search Tree
2) Repeat by reading student records from a file. Implement a menu driven program to create a phone book for student records which are read from a file. The application must have an option to search for a record based on roll no of the student. Search is done using a Binary Search Tree

---

REFERENCES

1. Behrouz A. Forouzan, Richard F. Gilberg "*A Structured Programming Approach Using C*", 3rd Edition, Cengage Learning India Private Limited, India, 2007.
2. Ellis Horowitz, Sartaj Sahni , Susan Anderson and Freed, " *Fundamentals of Data Structures in C*", 2nd Edition, Universities Press, India, Reprint 2011.
3. Richard F. Gilberg, Behrouz A. Forouzan, "*Data structures, A Pseudocode Approach with C*", 2nd Edition, Cengage Learning India Pvt. Ltd, India, 2009.
4. Robert Kruc & Bruce Lening, "*Data structures & Program Design in C*", 2nd Edition, Pearson, 2007.
5. Debasis Samanta, "*Classic Data Structures*", 2nd Edition, PHI Learning Pvt. Ltd., India, 2010

DEBUGGING A SAMPLE C PROGRAM WITH ERRORS:

Create the following C program using commands explained earlier.

```
1   # include <stdio.h>
2   int main()
3  {
4            int i, num, j;
5            printf ("Enter the number: ");
6            scanf ("%d", &num );
7            for (i=1; i<num; i++)
8                        j=j*i;
9            printf("The factorial of %d is %d\n",num,j);
10       }

$ cc factorial.c
$ ./a.out
Enter the number: 3
The factorial of 3 is 12548672
```

Now debug it while reviewing the most useful commands in gdb.

Step 1: Compile the C program with debugging option -g
Compile your C program with -g option. This allows the compiler to collect the debugging information.

```
$ cc -g factorial.c
```

Note: The above command creates a.out file which will be used for debugging as shown below.

Step 2: Launch gdb
Launch the C debugger (gdb) as shown below.

```
$ gdb a.out
```

Step 3: Set up a break point inside C program

```
Syntax:
```

```
break line_number
```

Other formats:

- break [file_name]:line_number
- break [file_name]:func_name

Places break point in the C program, where you suspect errors. While executing the program, the debugger will stop at the break point, and gives you the prompt to debug. So before starting up the program, let us place the following break point in our program.

```
break 8
Breakpoint 1 at 0x804846f: file factorial.c, line 8.
```

Step 4: Execute the C program in gdb debugger

```
run [args]
```

You can start running the program using the run command in the gdb debugger. You can also give command line arguments to the program via run args. The example program we used here does not requires any command line arguments so let us give run, and start the program execution.

```
run
Starting program: /home/student/Debugging/c/a.out
```

Once you executed the C program, it would execute until the first break point, and give you the prompt for debugging.

```
Breakpoint 1, main () at factorial.c:8
8                                   j=j*i;
```

You can use various gdb commands to debug the C program as explained in the sections below.

Step 5: Printing the variable values inside gdb debugger

```
Syntax: print {variable}
```

Examples:

```
print i
print j
print num
(gdb) p i
$1 = 1
(gdb) p j
$2 = 3042592
(gdb) p num
$3 = 3
(gdb)
```

As you see above, in the factorial.c, we have not initialized the variable j. So, it gets garbage value resulting in a big numbers as factorial values.

Fix this issue by initializing variable j with 1, compile the C program and execute it again.

Even after this fix there seems to be some problem in the factorial.c program, as it still gives wrong factorial value.

So, place the break point in 8th line, and continue as explained in the next section.

Step 6. Continue, stepping over and in – gdb commands

There are three kind of gdb operations you can choose when the program stops at a break point. They are continuing until the next break point, stepping in, or stepping over the next program lines.

- c or continue: Debugger will continue executing until the next break point.
- n or next: Debugger will execute the next line as single instruction.
- s or step: Same as next, but does not treats function as a single instruction, instead goes into the function and executes it line by line.

By continuing or stepping through you could have found that the issue is because we have not used the <= in the 'for loop' condition checking. So changing that from < to <= will solve the issue.

gdb command shortcuts

Use following shortcuts for most of the frequent gdb operations.

- l – list
- p – print

- c – continue
- s – step
- ENTER: pressing enter key would execute the previously executed command again.

Miscellaneous gdb commands

- **l command:** Use gdb command l or list to print the source code in the debug mode. Use l line-number to view a specific line number (or) l function to view a specific function.
- **bt: backtrack** – Print backtrace of all stack frames, or innermost COUNT frames.
- **help** – View help for a particular gdb topic — help TOPICNAME.
- **quit** – Exit from the gdb debugger.

C QUICK REFERENCE

PREPROCESSOR

```
// Comment to end of line
/* Multi-line
   Comment
 */
#include <stdio.h>              // Insert standard header file
#include "myfile.h"             // Insert file in current directory
#define X some text             // Replace X with some text
#define F(a,b) a+b              // Replace F(1,2) with 1+2
#define X \
        some text               // Line continuation
#undef X                        // Remove definition
#if defined(X)                  // Condional compilation (#ifdef X)
#else                           // Optional (#ifndef X or #if !defined(X))
#endif                          // Required after #if, #ifdef
```

LITERALS

```
255, 0377, 0xff                 // Integers (decimal, octal, hex)
2147463647L, 0x7fffffffl        // Long (32-bit) integers
123.0, 1.23e2                   // double (real) numbers
‘a’, ‘\141’, ‘\x61’             // Character (literal, octal, hex)
‘\n’, ‘\\’, ‘\’’, ‘\”’,         // Newline, backslash, single quote, double quote
“string\n”                      // Array of characters ending with newline and \0
“hello” “world”                 // Concatenated strings
```

DECLARATIONS

```
int x;                          // Declare x to be an integer (value undefined)
int x=255;                      // Declare and initialize x to 255
short s; long l;                // Usually 16 or 32 bit integer (int may be either)
char c= ‘a’;                    // Usually 8 bit character
unsigned char u=255; signed char m=-1; // char might be either
unsigned long x=0xffffffffL; // short, int, long are signed
float f; double d;              // Single or double precision real (never unsigned)
int a, b, c;                    // Multiple declarations
int a[10];                      // Array of 10 ints (a[0] through a[9])
int a[]={0,1,2};                // Initialized array (or a[3]={0,1,2}; )
int a[2][3]={{1,2,3},{4,5,6}; // Array of array of ints
```

```
char s[]= “hello”;              // String (6 elements including ‘\0’)
int* p;                         // p is a pointer to (address of) int
char* s= “hello”;               // s points to unnamed array containing "hello"
void* p=NULL;                   // Address of untyped memory (NULL is 0)
int& r=x;                       // r is a reference to (alias of) int x
enum weekend {SAT, SUN};// weekend is a type with values SAT and SUN
enum weekend day;               // day is a variable of type weekend
enum weekend {SAT=0,SUN=1}; // Explicit representation as int
enum {SAT,SUN} day;             // Anonymous enum
typedef String char*;           // String s; means char* s;
const int c=3;                  // Constants must be initialized, cannot assign
const int* p=a;                 // Contents of p (elements of a) are constant
int* const p=a;                 // p (but not contents) are constant
const int* const p=a;           // Both p and its contents are constant
const int& cr=x;                // cr cannot be assigned to change x
```

STORAGE CLASSES

```
int x;                          // Auto (memory exists only while in scope)
static int x;                   // Global lifetime even if local scope
extern int x;                   // Information only, declared elsewhere
```

STATEMENTS

```
x=y;                            // Every expression is a statement
int x;                          // Declarations are statements
;                               // Empty statement
{                               // A block is a single statement
   int x;                       // Scope of x is from declaration to end of
//block
   a;                           // In C, declarations must precede statements
}
if (x) a;                       // If x is true (not 0), evaluate a
else if (y) b;                  // If not x and y (optional, may be repeated)
else c;                         // If not x and not y (optional)
while (x) a;                    // Repeat 0 or more times while x is true
for (x; y; z) a;                // Equivalent to:  x; while(y) {a; z;}
do a; while (x);                // Equivalent to:  a; while(x) a;
switch (x) {                    // x must be int
        case X1: a;             // If x == X1 (must be a const), jump here
        case X2: b;             // Else if x == X2, jump here
        default: c;             // Else jump here (optional)
```

```
}
break;                  // Jump out of while, do, for loop, or switch
continue;               // Jump to bottom of while, do, or for loop
return x;               // Return x from function to caller
```

FUNCTIONS

```
int f(int x, int);      // f is a function taking 2 ints and returning int
void f();               // f is a procedure taking no arguments
void f(int a=0);        // f() is equivalent to f(0)
f();                    // Default return type is int
inline f();             // Optimize for speed
f( ) { statements; }    // Function definition (must be global)
```

Function parameters and return values may be of any type. A function must either be declared or defined before it is used. It may be declared first and defined later. Every program consists of a set of global variable declarations and a set of function definitions (possibly in separate files), one of which must be:

```
int main() { statements... }      or
int main(int argc, char* argv[]) { statements... }
```

argv is an array of argc strings from the command line. By convention, main returns status 0 if successful, 1 or higher for errors.

EXPRESSIONS

Operators are grouped by precedence, highest first.
Unary operators and assignment evaluate right to left.
All others are left to right.
Precedence does not affect order of evaluation which is undefined. There are no runtime checks for arrays out of bounds, invalid pointers etc.

```
t.x             // Member x of struct or class t
p → x           // Member x of struct or class pointed to by p
a[i]            // i'th element of array a
f(x, y)         // Call to function f with arguments x and y
x++             // Add 1 to x, evaluates to original x (postfix)
x--             // Subtract 1 from x, evaluates to original x
sizeof x        // Number of bytes used to represent object x
sizeof(T)       // Number of bytes to represent type T
++x             // Add 1 to x, evaluates to new value (prefix)
--x             // Subtract 1 from x, evaluates to new value
```

```
~x                  // Bitwise complement of x
!x                  // true if x is 0, else false (1 or 0 in C)
-x                  // Unary minus
+x                  // Unary plus (default)
&x                  // Address of x
*p                  // Contents of address p (*&x equals x)
x * y               // Multiply
x / y               // Divide (integers round toward 0)
x % y               // Modulo (result has sign of x)
x + y               // Add, or &x[y]
x – y               // Subtract, or number of elements from *x to *y
x << y              // x shifted y bits to left (x * pow(2, y))
x >> y              // x shifted y bits to right (x / pow(2, y))
x < y               // Less than
x <= y              // Less than or equal to
x > y               // Greater than
x >= y              // Greater than or equal to
x == y              // Equals
x != y              // Not equals
x & y               // Bitwise and (3 & 6 is 2)
x ^ y               // Bitwise exclusive or (3 ^ 6 is 5)
x | y               // Bitwise or (3 | 6 is 7)
x && y              // x and then y (evaluates y only if x (not 0))
x || r              // x or else y (evaluates y only if x is false(0))
x = y               // Assign y to x, returns new value of x
x += y              // x = x + y, also -= *= /= <<= >>= &= |= ^=
x ? y : z           // y if x is true (nonzero), else z
x, y                // evaluates x and y, returns y (seldom used)
```

STDIO.H

```
prinf("Format specifers",value1,value2,..);
printf("user defined message");
scanf("Format specifiers",&value1,&value2,.....);
```

Format specifier:

Format specifier	Type of value
%d	Integer
%f	Float
%lf	Double
%c	Single character
%s	String
%u	Unsigned int
%ld	Long int
%lf	Long double

STRING Handling functions

```
strcat(s1,s2)              // Concatenates s2 to s1
strcmp(s1,s2)              // Comparison of two strings and returns 0 if equal
strstr(mainstr, substr)    // Search for the Substring in mainstr
strlen(s1)                 // string length of s1
gets(s1)                   // Read line ending in '\n'
```

Other functions

```
asin(x); acos(x); atan(x);     // Inverses
atan2(y, x);                   // atan(y/x)
sinh(x); cosh(x); tanh(x);     // Hyperbolic
exp(x); log(x); log10(x);      // e to the x, log base e, log base 10
pow(x, y); sqrt(x);            // x to the y, square root
ceil(x); floor(x);             // Round up or down (as a double)
fabs(x); fmod(x, y);           // Absolute value, x mod y
```

FILE HANDLING IN C

Before we read (or write) information from (to file) on a disk, we must open a file in particular mode. To open a file we use fopen() function. fopen() is a high level IO function. There are many high level IO function we use in file handling.

Syntax:

```
FILE *fp;
fp= fopen("filename","mode");
```

FILE is a type supported in 'C'.
fp = this is a file pointer of type FILE data structure.
fopen = High level IO function
filename = valid filename which you want to open
mode = r for read, w for write, a for append

I. **Binary Files:** To read, write contents of a binary file we use fread and fwrite respectively. Examples of binary files are .exe files .bmp or image files. More generally, any file which cannot be read using any normal text editor.

Functions to read contents: fread() and fwrite()

Syntax: size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

The functions fread/fwrite are used for reading/writing data from/to the file opened by fopen function. These functions accept three arguments. The first argument is a pointer to buffer used for reading/writing the data. The data read/written is in the form of 'nmemb' elements each 'size' bytes long.

Reading a particular byte in a file: fseek()

Syntax: int fseek(FILE *stream, long offset, int whence);

The fseek() function is used to set the file position indicator for the stream to a new position. This function accepts three arguments. The first argument is the FILE stream pointer returned by the fopen() function. The second argument 'offset' tells the amount of bytes to seek. The third argument 'whence' tells from where the seek of 'offset' number of bytes is to be done. The available values for whence are

SEEK_SET, SEEK_CUR, or SEEK_END. These three values (in order) depict the start of the file, the current position and the end of the file. Upon success, this function returns 0, otherwise it returns -1.

Closing a file: fclose()

Syntax: int fclose(FILE *fp);

The fclose() function first flushes the stream opened by fopen() and then closes the underlying descriptor. Upon successful completion this function returns 0 else end of file (eof) is returned. In case of failure, if the stream is accessed further then the behavior remains undefined.

Writing into binary file:

Example: To write an employ's record to a file. Record has three fields name age and basic salary.

```
struct emp{
  char name[20];
  int age;
  float bs;
};
struct emp e;
void main(){
        fp = fopen("EMP.DAT","wb");
        if(fp == NULL){
        printf("Cannot open file");
        }
        printf("Enter name, age and basic salary");
        scanf("%s%d%f",e.name,&e.age,&e.bs);
        fwrite(&e,sizeof(e),1,fp);
        fclose(fp);
}
```

Reading from the binary file:

The following code segment illustrates how to read data from file in binary format.

```
struct emp{
  char name[20];
  int age;
```

```
  float bs;
};
struct emp e;
void main(){
        fp = fopen("EMP.DAT","rb");
        /* read all the records one by one */
        while(fread(&e,sizeof(e),1,fp)==1){
          printf("%s %d %f\n",e.name,e.age,e.bs);
        }
        fclose(fp);
}
```

II. Text File: File contents are written in ascii or unicode format. Generally, these files can be read using a text editor.

Writing to text file: fprintf()

Syntax: fprintf(FILE*,format,variable_list);

The fprintf statement should look very familiar to you. It can be almost used in the same way as printf. The only new thing is that it uses the file pointer as its first parameter.

```
#include<stdio.h>
int main()
{
        FILE *ptr_file;
        int x;

        ptr_file =fopen("output.txt", "w");

        if (ptr_file == NULL)
                return -1;

        for (x=1; x<=10; x++)
                fprintf(ptr_file,"%d\n", x);

        fclose(ptr_file);
}
```

Reading a text file: fscanf()

Syntax: int fscanf(FILE *stream, const char *format, variable_list)
The C library function int fscanf(FILE *stream, const char *format, ...) reads formatted input from a stream.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        char str1[10], str2[10], str3[10];
        int year;
        FILE * fp;
        fp = fopen ("file.txt", "w+");
        fputs("We are in 2012", fp);
        rewind(fp);
        fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
        printf("Read String1 |%s|\n", str1 );
        printf("Read String2 |%s|\n", str2 );
        printf("Read String3 |%s|\n", str3 );
        printf("Read Integer |%d|\n", year );
        fclose(fp);
        return(0);
}
```

Similar to fprintf, fscanf, we can use fputc, fgetc to handle character data.