Lab1 and Lab2 (Refer DSA lab 2 instructions, DSA lab 3 and 4 instructions videos)

SOLVING PROBLEMS USING ARRAYS, POINTERS AND DYNAMIC MEMORY ALLOCATION FUNCTIONS

Objectives:

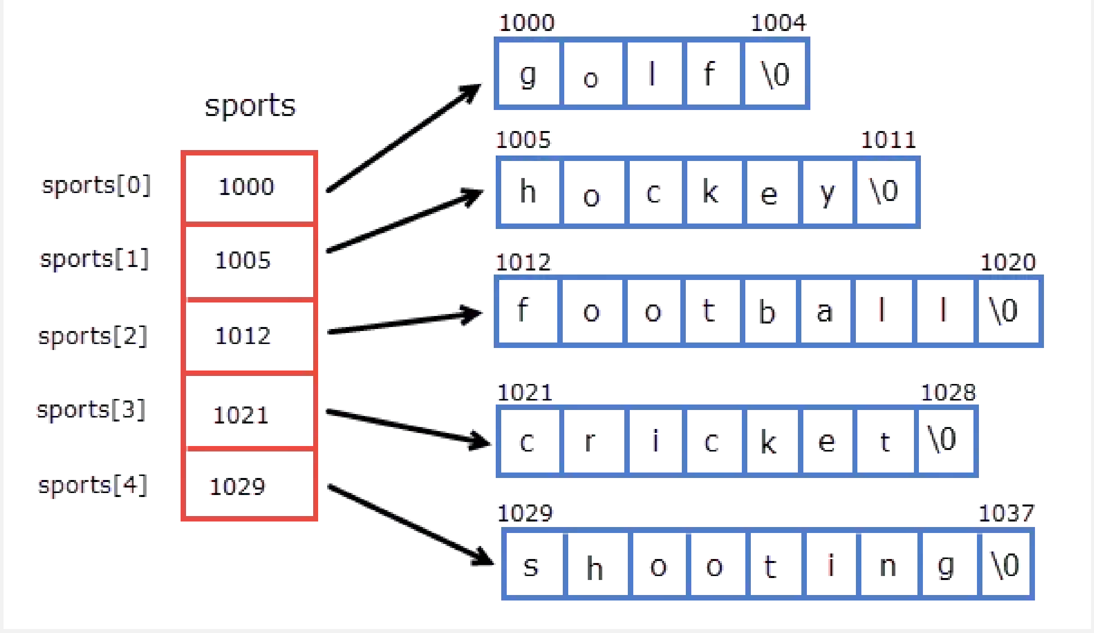
In this lab, student will be able to:
i) Familiarize with syntax and usage of pointers and pointer to arrays and dynamic memory management functions
ii) Write C programs making use of pointer concepts and dynamic memory allocation functions

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;
}
```

SOLVING PROBLEMS USING RAGGED ARRAYS, STRUCTURES AND POINTERS

Objectives:

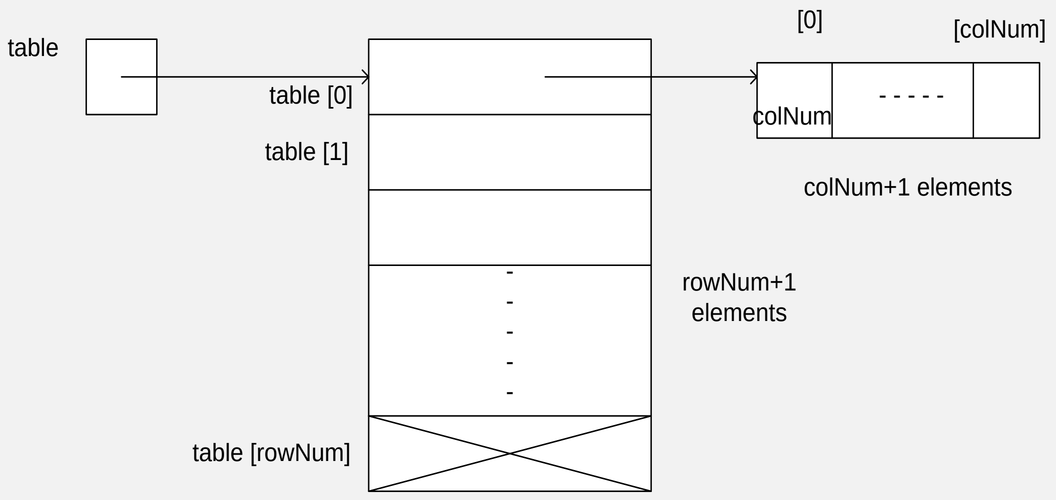
In this lab, student will be able to:

- 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
```

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
```

Questions for LAB1

1. 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.
2. 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.
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.

Note : You may use the following structure .

```
struct DOB {
   int day, month, year;
}
struct ADRS {
   int house_no;
   long zipcode;
   char state[20];
}
struct EMPLOYEE {
   char name[20];
   struct DOB dob;
   struct ADRS address;
}
struct EMPLOYEE emp[10];
EMPLOYEE* ptr = emp;
```

Questions for LAB2

1. 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.

Note: You may use the following structure.

```
struct DOB{
   int day;
   char* mth;
   int year;
}
struct STU_INFO{
   int reg_no;
   char* name;
   char[20] adrs;
}
struct COLLEGE{
   char* clg_name;
   char[20] univ_name;
}
struct STUDENT{
   struct DOB dob;
   struct STU_INFO stu_info;
   struct COLLEGE clg;
}

struct STUDENT student;
char[10] month;
```

```
scanf(“%s”, month);
student.dob.mth = (char*) malloc (sizeof (month);
strcpy(student.dob.mth, month);
```

2. Write C programs using recursion to copy one string to another using Recursion.
3. Write C programs using recursion to check whether a given String is Palindrome or not, using Recursion

4. Write C programs using recursion to simulate the working of Tower of Hanoi for n disks. Print the number of moves.