Experiment 1 **Date: 21.09.2023**

Advanced Use of GCC

Aim

1. Advanced Use of GCC Aim: 1. Advanced use of gcc : Important Options -o, -c, -D, -l, -I, -g, -O, -save-temps, -pg

Write a C program 'sum.c' to add two numbers. Read the input from Standard Input and write output to Standard output. Compile and generate output using gcc command and its important options.

Program

```
#include <stdio.h>
 int main() {
 int a,b,sum;
 printf("Enter the first number: ");
 scanf("%d", &a)
 printf("Enter the second number: ");
 scanf("%d", &b)
 sum = a + b;
 printf("Sum of %d and %d is %d\n", a, b, sum)
 return 0;
}
```

GCC

GCC is a Linux-based c compiler released by the free software foundation which is usually operated via the command line. It often comes distributed freely with a Linux installation, so if you are running Unix or a Linux variant you will probably have it on your system. You can invoke gcc on a source code file simply by typing:-

gcc filename

The default executable output of gcc is "a.out", which can be run by typing"./a.out". It is also possible to specify a name for the executable file at the command line by using the syntax " -o outputfile", as shown in the following example: -

gcc filename -o outputfile

Again, you can run your program with "./outputfile". (the ./ is there to ensure to run the program for the current working directory.)

Note: if you need to use functions from the math library (generally functions from math.h" such as sin or sqrt), then you need to explicitly ask it to link with that library with the "-1" flag and the library "m":

gcc filename -o outputfile -lm

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc sum.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out sum.c
```

Enter first number : 10

Enter Second number: 20

Sum of 10 and 20 is : 30

Important Options in GCC

Option: -o
To write and build output to output file.
Output
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc sum.c -o sum_out

Here, GCC compiles the sum.c file and generates an executable namedsum_out.
Option: -c
To compile source files to object files without linking.
Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -c sum.c
This will generate an object file sum.o that can be linked separately.

Option: -D
To define a preprocessor macro.
Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -D debug=1 sum.c
This defines the macro 'DEBUG' with the value 1, which can be used inthe source code.
Option: -l
To include a directory of header files.
Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -o sum.c sum_out.c -lmHere, the -lm option links the math library (libm) with the sum.c.
Option: -I
To look in a directory for library files.
Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -o sum.c sum_out.c -I./ads_lab
This tells GCC to look for header files in the ads_lab directory.
Option: -g
To debug the program using GDB.
Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -g sum.c -o sum_out
This compiles sum.c with debug information, enabling you to debug theresulting

executable.

Option: -O

To optimize for code size and execution time.

Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -O3 -o my_pgm sum.cThis compiles sum.c with a high level of optimization.

Option: -pg

To enable code profiling.

Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -pg -o my_pgm source.c
This compiles source.c with profiling support, allowing you to useprofilers like gprof.

Option: -save-temps

To save temporary files generated during program execution.

Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -save-temps -o my_pgmsource.c
This will generate intermediate files, like sum.i (pre-processed source)and sum.s (assembly code), in addition to the final executable.

Experiment 2 Date: 21.09.2023

Familiarisation with GDB

Aim:

2. Familiarisation with gdb: Important Commands - break, run, next, print,display, help.
Write a C program 'mul.c' to multiply two numbers. Read the input from Standard Input and write output to Standard output. Compile and generate sum.out which is then debug with gdb and commands.

Program

```
#include <stdio.h>
 int main()
 {
int num1, num2, product;
printf("Enter the first integer: ");
scanf("%d", &num1);
 printf("Enter the second integer: ");
scanf("%d", &num2);
 product = num1 * num2;
 printf("Product of %d and %d is %d\n", num1, num2, product);
return 0;
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -g mul.c -o mul_out
mits@mits:~/Desktop/S1MCA/ADS_lab$ gdb mul_out

GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90 Copyright (C) 2022
Free Software Foundation, Inc.License GPLv3+: GNU GPL version
3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.There is NO
WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.This GDB was
configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.For bug
reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...Reading symbols from
sum1...
(gdb) run
```

Starting program: /home/mits/Desktop/S1MCA/sum1[Thread
debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-
gnu/libthread_db.so.1".

Enter 2 numbers : 10 20
Product : 200 [Inferior 1 (process 23588) exited normally](gdb) **quit**
Important Commands in GDB

Command: break

Sets a breakpoint on a particular line.
Output
(gdb) break mul.c:5

Command: run
Executes the program from start to end.
Output
(gdb) run
Command: next
Executes the next line of code without diving into functions.
Output
(gdb) next

Command: print
Displays the value of a variable.
Output
(gdb) print a(gdb)
a 10
Command: display
Displays the current values of the specified variable after every step.
Output
(gdb) display a

Experiment 3 **Date: 29.09.2023**

Familiarisation with gprof

Aim:

3. Write a program for finding the sum of two numbers using function. Thenprofile the executable with gprof.

Program

```
#include <stdio.h>
int sum(int num1, int num2) {
return num1 + num2;}
int main() {
        int num1, num2;
        printf("Enter the first number: ");
        scanf("%d", &num1);
        printf("Enter the second number: ");
       scanf("%d", &num2);
       int result = sum(num1, num2);
       printf("Sum of %d and %d is %d\n", num1, num2, result);
       return 0;
}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc sum.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc ./a.out sum.c
Enter first number : 40
Enter first number: 30
Sum of and 30 and 40 is 70
mits@mits:~/Desktop/S1MCA/ADS_lab$ gprof ./sum.out gmon.out >pgm3.txt


Flat profile:
Each sample counts as 0.01 seconds.
% cumulative self self total
time seconds seconds calls ms/call ms/call name
50.00 0.05 0.05 main
25.00 0.07 0.02 sum
[... More profiling data ...]
Call graph:
index % time self children called name
0.00 0.05 1/1 main [3]
[... Call graph details ...]
Index by function name:
[... Index details ...]
```

Experiment 4 Date: 29.09.2023

Different types of functions

Aim:

4. Write a program for finding the sum of two numbers using different typesof functions.

Algorithm:

main ()

1. Start
2. Declare choice, num1, num2, result
3. Display choices.
4. Read option into choices
   a. If choice==1 call sumWithoutArgsAndReturn ()
   b. If choice==2 Input num1 and num2 and call sumWithoutReturn (num1, num2)
   c. If choice==3 Store result=sumWithReturn () and print result
   d. If choice==4 Input num1, num2 then store Result=sumWithArgs (num1, num2) and Print sumWithArgs ()
   e. If choice=5 Exit
5. Repeat 3,4 while choice not equal to 5
6. Stop

sumWithoutArgsAndReturn ()

1. Start
2. Declare num1 and num2
3. Read num1, num2
4. Print num1+num2
5. Exit

sumWithoutReturn ()

1. Start
2. Print num1+num2
3. Exit

sumWithReturn ()

1. Start
2. Declare num1, num2, sum
3. Read num1, num2
3. sum=num1+num2
4. Return sum
5. Exit

sumWithArgs ()

1. Start
2. Return=num1+num2
3. Exit

Program

```
#include <stdio.h>
   void sumWithoutArgsAndReturn() {
```

```
    float num1, num2;
    printf("Enter two numbers: ");
    scanf("%f %f", &num1, &num2);
    printf("Sum: %f\n", num1 + num2);
}

void sumWithoutReturn(float num1, float num2) {
    printf("Sum: %f\n", num1 + num2);
}

float sumWithReturn() {
    float num1, num2, sum;
    printf("Enter two numbers: ");
    scanf("%f %f", &num1, &num2);
    sum = num1 + num2;
    return sum;
}

float sumWithArgs(float num1, float num2) {
    return num1 + num2;
}

int main() {
    int choice;
    float num1, num2,result;

    do {
        printf("\nMenu:\n");
        printf("1. Function without return type and arguments\n");
        printf("2. Function without return type and with arguments\n");
        printf("3. Function with return type and without arguments\n");
        printf("4. Function with return type and arguments\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                sumWithoutArgsAndReturn();
                break;
            case 2:
                printf("Enter two numbers: ");
                scanf("%f %f", &num1, &num2);
                sumWithoutReturn(num1, num2);
                break;
```

```
            case 3:
                result=sumWithReturn();
                printf("Sum: %f\n",result );
                break;
            case 4:
                printf("Enter two numbers: ");
               scanf("%f %f", &num1, &num2);
               result=sumWithArgs(num1, num2)
               printf("Sum: %f\n",result );
                break;
            case 5:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice.\n");
        }
    }
        while (choice != 5);

    return 0;
     }
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc Functions.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out
Menu:
1. Function without return type and arguments
2. Function without return type and with arguments
3. Function with return type and without arguments
4. Function with return type and arguments
5. Exit
Enter your choice: 1
Enter two numbers: 10 20
Sum: 30.000000
 Menu:
1. Function without return type and arguments
2. Function without return type and with arguments
3. Function with return type and without arguments
4. Function with return type and arguments
5. Exit
Enter your choice: 2
Enter two numbers: 90 100
Sum: 190.000000
Menu:
```

1. Function without return type and arguments
2. Function without return type and with arguments
3. Function with return type and without arguments
4. Function with return type and arguments
5. Exit
Enter your choice: 3
Enter two numbers: 40 55
Sum: 95.000000

Menu:
1. Function without return type and arguments
2. Function without return type and with arguments
3. Function with return type and without arguments
4. Function with return type and arguments
5. Exit
Enter your choice: 4
Enter two numbers: 60 80
Sum: 140.000000
Menu:
1. Function without return type and arguments
2. Function without return type and with arguments
3. Function with return type and without arguments
4. Function with return type and arguments
5. Exit
Enter your choice: 5
Exiting program.

Experiment 5 Date:06.10.2023

Array Operations

Aim:

To implement a menu driven program to perform following array operations

i. Insert an element to a particular location

ii. Delete an element from a particular location

iii. Traverse

Algorithm:

main()

1. Start
2. Declare an array[MAX_SIZE] and initialize size to 0 as global.
3. Declare choice.
4. Repeat until choice is 4:
5. Display a menu:
   1. Insert an element 2.Delete an element 3. Traverse the array 4.Exit
6. Read option into choice
   a. If choice is 1, call insertElement()
   b. If choice is 2, call deleteElement()
   c. If choice is 3, call traverseArray()
   d. If choice is 4, exit
7. Stop.

insertElement()

1. Declare element, position.
2. Read position.
3. if (position < 0 || position > size) {
   Print Invalid position
4. Repeat for (int i = size; i > position; i--):
5. Set array[i] to array[i - 1].
6. Set array[position] to element.
7. Increment size.
8. Exit

deleteElement()

1. Start
2. Declare position.
3. Read position.
4. if (position < 0 || position >= size)
   Print "Invalid position.
5. Repeat for (int i = position; i < size - 1; i++):
6. Set array[i] to array[i + 1].
7. Decrement n.
8. Exit

traverseArray()

1. Start
2. Repeat for (int i = 0; i <size; i++):
3. Print array[i].
4. Exit

Program

```
#include <stdio.h>
#define MAX_SIZE 100
int array[MAX_SIZE];
int size = 0;
void insertElement() {
    int position, element;
    if (size >= MAX_SIZE) {
        printf("Array is full. Cannot insert more elements.\n");
        return;
    }
    printf("Enter the position to insert: ");
    scanf("%d", &position);
    if (position < 0 || position > size) {
        printf("Invalid position. Please enter a valid position.\n");
        return;
    }
  for (int i = size; i > position; i--) {
        array[i] = array[i - 1];
    }
    printf("Enter the element to insert: ");
    scanf("%d", &element);
    array[position] = element;
    size++;
 printf("Element inserted successfully.\n");
}
void deleteElement() {
    int position;
    if (size <= 0) {
        printf("Array is empty. Cannot delete any element.\n");
        return;
    }
    printf("Enter the position to delete: ");
    scanf("%d", &position);
    if (position < 0 || position >= size) {
        printf("Invalid position. Please enter a valid position.\n");
        return;
    }
    for (int i = position; i < size - 1; i++) {
```

```
        array[i] = array[i + 1];
    }
    size--;
    printf("Element deleted successfully.\n");
}
void traverseArray() {
    if (size <= 0) {
        printf("Array is empty.\n");
        return;
    }
    printf("Array elements: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }
   printf("\n");
}
int main() {
   int choice;
   do {
        printf("\nMenu:\n");
        printf("1. Insert an element\n");
        printf("2. Delete an element\n");
        printf("3. Traverse the array\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
       scanf("%d", &choice);
       switch (choice) {
            case 1:
                insertElement();
                break;
            case 2:
             deleteElement();
                break;
            case 3:
               traverseArray();
                break;
            case 4:
                printf("Exiting program.\n")
  break;
                default:
                printf("Invalid choice. Please enter a valid option.\n");
        }


      } while (choice != 4);
     return 0;
```

```
}
```

OUTPUT

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc array.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out

Menu:
1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 1
Enter the position to insert: 0
Enter the element to insert: 2
Element inserted successfully.

Menu:
1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 1
Enter the position to insert: 1
Enter the element to insert: 5
Element inserted successfully.

Menu:
1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 1
Enter the position to insert: 2
Enter the element to insert: 3
Element inserted successfully.


Menu:
1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 3
Array elements: 2 5 3
Array elements: 2 5 3
Menu:
```

1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 2
Enter the position to delete: 2
Element deleted successfully.

Menu:
1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 2
Enter the position to delete: 1
Element deleted successfully.

Menu:
1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 3
Array elements: 2

Menu:
1. Insert an element
2. Delete an element
3. Traverse the array
4. Exit
Enter your choice: 4
Exiting program.

Experiment 6 Date: 06.10.2023

sort an integer array

Aim:

6.Program to sort an integer array

Algorithm:

main ()

1. Declare n and arr[n]
2. Read n
3 for (int i=0;i<n;i++)
4. Print arr[i].
5. Call bubbleSort(arr, n)
6. Print Sorted array
7. for (int i=0;i<n;i++)
8. Print arr[i].
9. Stop

bubbleSort(int arr[], int n)

1. Start
2. for (int i = 0; i < n - 1; i++)
   for (int j = 0; j < n - i - 1; j++)
   if (arr[j] > arr[j + 1])
   Swap arr[j] and arr[j + 1]
3. Set temp = arr[j];
4. Set arr[j] = arr[j + 1];
5. Set arr[j + 1] = temp;
6. Exit

PROGRAM

```
#include <stdio.h>
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }}
    }}
int main() {
    int n;
    printf("Enter the number of elements in the array: ");
```

```
    scanf("%d", &n);
    int arr[n];
    printf("Enter the elements of the array:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
   printf("Original array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    bubbleSort(arr, n);
    printf("\nSorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

OUTPUT

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc sort.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out

Enter the number of elements in the array: 4
Enter the elements of the array:
21 33 12 56
Original array: 21 33 12 56
Sorted array: 12 21 33 56
```

Experiment 7 **Date: 06.10.2023**

linear search and binary search

Aim:

7.Program to implement linear search and binary search

Algorithm:

main()

1. Start
2. Declare a[100],n,i,s,choice
3. Input n,s
4. for i=0 to n do
   {
   input a[i]
   set i=i+1
   }
5. Display 1.Linear search 2.Binary search Exit
6. Read option into choices.
   a.If choice==1 then
   call linearSearch(a,n,s)b.
   b.If choice==2 then
   call bublesort(a,n)
   call binarySearch(a,n,s)
7. Repeat 5,6 while ch not equal 3
8. stop

void bublesort(int a[],int n)

1. Start
2. Declare temp
3. for i=0 to n-1 do
   {
   for j=0 to n-i-1 do
   {
   if(a[j]>a[j+1])then
   {
   Set temp=a[j]
   Seta[j]=a[j+1]
   Set a[j+1]=temp
   }}
   }
4. exit

void linearSearch(int a[], int n, int s)

1. Start
2. Declare and initialize i,f=0
3. for i=0 to n do
   {
      if (a[i] == s) then
      {
         Set f =
         1Print i
      }
   }
4. if (f == 0) then
      Print 'Element not found'
5. exit

Void binarySearch (int a[], int n, int s)

1. start
2. Declare and initialize l = 0, u = n - 1, pos = -1, mid
3. while(l<=u) do
   {
Set mid = (l + u)/2;
if (s == a[mid]) then
      {
            Set pos = mid
            break
      }
      else if (a[mid] > s)
            set u = mid – 1
      else
            set l = mid + 1
   }
4. if (pos == -1) then
      Print 'Element not found'else then
      Print pos
5. exit

PROGRAM

```
#include <stdio.h>
void bublesort(int a[],int n)
{
 for(int  i=0;i<n-1;++i)
 {
```

```
        for(int  j=0;j<n-i-1;++j)
        {
                if(a[j]>a[j+1])
                {
                        int
                        temp=a[j];
                        a[j]=a[j+1];
                        a[j+1]=temp
                        ;
                }
        }
  }
}
void linearSearch(int a[], int n, int s)
{
 int i, f = 0;
 for (i = 0; i < n; ++i)
 {
        if (a[i] == s)
        {
                f = 1;
                printf("Element present on index:
                %d\n", i);break;
        }
 }
 if (f == 0)
   printf("Element not found\n");
}

void binarySearch(int a[], int n, int s)
{
 int l = 0, u = n - 1, pos = -1,
  mid;while (l <= u)
  {
        mid = (l + u) /2;
        if (s == a[mid])
        {
                pos =mid;
                break;
        }
       else if (a[mid] > s)
                u = mid – 1;
       else

              l = mid + 1;

         if (pos == -1)
```

```
  printf("Element not found\n");
   else
  printf("Element on index: %d\n", pos);
  }
 int main()
   {
     int a[100], n, choice, s;
     printf("Enter limit: ");
     scanf("%d", &n);
     printf("Enter
     elements:");for (int i =
     0; i < n; ++i)
     scanf("%d",&a[i]);
     while (choice!=3)
      {
printf("\nMenu:\n");
printf("1. Linear Search\n");
printf("2. Binary Search\n");
printf("3. Exit\n");
printf("Enter your choice");
scanf("%d", &choice);
switch (choice)
{
case 1:printf("Enter element to search: ");
scanf("%d", &s);
linearSearch(a, n, s);break;
case 2: bublesort(a,n);
printf("Enter element to search: ");
scanf("%d", &s);
binarySearch(a, n, s);
break;
default: printf("Invalid choice, please try again.\n");
       }
     }
     return 0;
  }
```

OUTPUT

```
Enter limit :3
Enter elements:7 5 3
Menu:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice:1
Enter element to search: 7
Element present on index: 0
Menu: 1.
Linear Search
2. Binary Search
3. Exit
Enter your choice: 2
Enter element to search: 5
Element on index: 1
Menu:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice: 1
Enter element to search: 6
Element not found.
```

Experiment 8 Date: 06.10.2023

Matrix Operations

Aim:

8. Perform addition, subtraction and multiplication of two matrices using switch.

Algorithm

main ()

1. Start
2. Declare matrix1[m][n], matrix2[m][n], result[m][n];
3. Declare integers m, n, choice
5. Repeat for each row i from 0 to m - 1:
   Repeat for each column j from 0 to n - 1:
6. Print matrix1[i][j]
   a. Repeat for each row i from 0 to m - 1:
   b. Repeat for each column j from 0 to n – 1
7. Print matrix2[i][j]
8. Repeat until choice is 4:
   a. Display a menu for matrix operations:
      1. Addition 2. Subtraction 3. Multiplication 4. Exit
9. Read choice into choice
   i. If choice is 1, call matrixAddition()
   ii. If choice is 2, call matrixSubtraction()
   iii. If choice is 3, call matrixMultiplication()
   iv. If choice is 4, exit

matrixAddition()

1. Start
2. Declare integer i, j
3. Repeat for each row i from 0 to m - 1:
   a. Repeat for each column j from 0 to n - 1:
      i. Set result[i][j] to matrix1[i][j] + matrix2[i][j]
4. Display the result matrix
5.Exit

matrixSubtraction()

1. Start
2. Declare integer i, j
3. Repeat for each row i from 0 to m - 1:
   a. Repeat for each column j from 0 to n - 1:
      i. Set result[i][j] to matrix1[i][j] - matrix2[i][j]

4. Display the result matrix
5.Exit

matrixMultiplication()
1. Start
2. Declare integer i, j, k
3. Repeat for each row i from 0 to m - 1:
   a. Repeat for each column j from 0 to n - 1:
      i. Set result[i][j] to 0
      ii. Repeat for each k from 0 to n - 1:
         iii. Set result[i][j] to result[i][j] + matrix1[i][k] * matrix2[k][j]
4. Display the result matrix
5.Exit

Program

```
#include <stdio.h>
int main() {
    int m, n, i, j;
    printf("Enter the number of rows and columns for the matrices: ");
    scanf("%d %d", &m, &n);
  int matrix1[m][n], matrix2[m][n], result[m][n];
    printf("Enter elements of matrix1:\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }
    printf("Enter elements of matrix2:\n");
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }
  int choice;
    do {
        printf("\nChoose operation:\n");
        printf("1. Addition\n2. Subtraction\n3. Multiplication\n");
        printf("4. Exit\n");
        scanf("%d", &choice);
        switch (choice) {
```

```
        case 1:
            for (i = 0; i < m; i++) {
                for (j = 0; j < n; j++) {
                    result[i][j] = matrix1[i][j] + matrix2[i][j];
                }
            }
            printf("\nResult of addition:\n");
            break;
        case 2:
            for (i = 0; i < m; i++) {
                for (j = 0; j < n; j++) {
                    result[i][j] = matrix1[i][j] - matrix2[i][j];
                }
            }
            printf("\nResult of subtraction:\n");
            break;
        case 3:
            for (i = 0; i < m; i++) {
                for (j = 0; j < n; j++) {
                    result[i][j] = 0;
                    for (int k = 0; k < n; k++) {
                        result[i][j] += matrix1[i][k] * matrix2[k][j];
                    }
                }
            }
            printf("\nResult of multiplication:\n");
            break;
     case 4:
            printf("Exiting the program.\n");
            break;
    default:
            printf("Invalid choice\n");
            break;
     }
     if (choice != 0) {
        for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++) {
                printf("%d ", result[i][j]);
            }
            printf("\n");
```

```
        }}}
       while (choice != 0);
  return 0;}
```

Output

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc matrix.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out

Enter the number of rows and columns for the matrices: 2 2
Enter elements of matrix1:
1 3 5 4
Enter elements of matrix2:
4 5 3 1
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
4. Exit
1
Result of addition:
5 8
8 5
Choose operation:
1. Addition
2. Subtraction
3. Multiplication
4. Exit
2
Result of subtraction:
-3 -2
2  3

Choose operation:
1. Addition
2. Subtraction
3. Multiplication
4. Exit
3
Result of multiplication:
```

13 8
32 29

Choose operation:
1. Addition
2. Subtraction
3. Multiplication
4. Exit
4
Exiting the program.

Experiment 9 Date: 12.10.2023

STACK OPEARATION USING ARRAY

Aim:

9. Program to implement stack operation using array

Algorithm

main()

1. Start
2. Declare stack[100], choice, n, top, item, i
3. Initialize top to -1
4. Read n
5. Display "STACK OPERATIONS:"
6. Display "1. PUSH\n2. POP\n3. DISPLAY\n4. EXIT"
7. Repeat until choice is 4:
8. Read choice into choice

    i. Case 1: Call push()
    ii. Case 2: Call pop()
    iii. Case 3: Call display()
    iv. Case 4: Display "EXIT"
    v. Default: Display "Enter a Valid Choice"
9. Stop

push()

1. Start
2. If top is greater than or equal to n - 1:
    a. Display "Overflow"
3. Else:
    a. Display "Enter a value to be pushed:"
    b. Read item into item
    c. Increment top
    d. Set stack[top] to item
4. Exit

pop()

1. Start
2. If top <=-1:
    a. Display "Underflow"
3. Else:
    a. Display "The element popped is " + stack[top]

b. Decrement top

display()

1. Start
2. If top >=0:
   a. Display "The elements in STACK:"
   b. Repeat for i from top to 0:
      i. Display stack[i]
   c. Display "Press Next Choice"
3. Else:
   Display "The STACK is empty"
4. Exit.

PROGRAM

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

```
            pop();
            break;
         }
         case 3:
         {
            display();
            break;
         }
         case 4:
         {
            printf("\n\t EXIT");
            break;
         }
         default:
         {
            printf ("\n\t Enter a Valid Choice");
         }}}
   while(choice!=4);
   return 0;
}
void push()
{
   if(top>=n-1)
   {
      printf("\n\t over flow");

   }
   else
   {
      printf(" Enter a value to be pushed:");
      scanf("%d",&item);
      top++;
      stack[top]=item;
   }
}
void pop()
{
   if(top<=-1)
   {
      printf("\n\t  under flow");
```

```
    }
    else
    {
        printf("\n\t The elements  popped is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n");
        for(i=top; i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }}
```

OUTPUT

```
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc stack.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out

Enter a value to be pushed:2
 Enter the Choice:1
 Enter a value to be pushed:4
 Enter the Choice:1
 Enter a value to be pushed:3
 Enter the Choice:1
 over flow
 Enter the Choice:3
 The elements in STACK
3
4
2
 Press Next Choice
 Enter the Choice:2
 The elements  popped is 3
```

```
Enter the Choice:2
The elements  popped is 4
Enter the Choice:2
The elements  popped is 2
Enter the Choice:2
under flow
Enter the Choice:3
The STACK is empty
Enter the Choice:4
Exit
```

Experiment 10 **Date: 12.10.2023**

Queue Operations

Aim:

10. Program to implement queue operations using arrays

Algorithm:

void main()

1. Start
2. int ch;
3. Enter the size.
4. Enter the choice.
5. switch(ch)
   case 1:
   insertion();
   display();
   break;

   case 2:
   delete();
   display();
   break();

   case 3:
   display();
   break;
6. Stop.

void insertion()

1. Start
2. Enter the element to insert
3. if (rear == size - 1)
   print “Overflow…”
4. else if (front == -1 && rear == -1)
   front = 0;
   rear = 0;
5. else
   rear = rear + 1;
6. q[rear] = ele;

7. print “Element inserted”
8. Stop

void delete()

1. Start
2. if (front == -1 && rear == -1)
   print “Underflow…”
3. else
   ele = q[front];
   if (front == rear)
   {
   front = -1;
   rear = -1;
   }
   else
   {
   front = front + 1;
   }
4. print “Element deleted”
5. stop

void display()

1. Start
2. if (rear == -1)
   print “Empty Queue”
3. else
   for (int i = front; i <= rear; i++)
   print q[i];
4. Stop

Program:

```
#include <stdio.h>

int q[10];
int size, ele;
int front = -1, rear = -1;

void insertion();
```

```
void delete();
void display();

int main() {
   int ch;
   printf("Enter the size:");
   scanf("%d", &size);
   printf("Operation available are\n1.Insertion\n2.Deletion\n3.Traverse");

   do{
      printf("\nEnter a choice");
      scanf("%d", &ch);
      switch (ch) {
         case 1: {
            insertion();
            display();
            break;
         }
         case 2: {
            delete();
            display();
            break;
         }
         case 3:
         {
            display();
            break;
         }
      }
   }while(ch<3);
}

void insertion()
{
   printf("Enter the element to insert:");
   scanf("%d", &ele);

   if (rear == size - 1)
   {
      printf("Overflow...\n");
   }
   else
   {
      if (front == -1 && rear == -1)
```

```
        {
            front = 0;
            rear = 0;
        }
        else
        {
            rear = rear + 1;
        }
        q[rear] = ele;
        printf("The element %d inserted\n", ele);
    }
}
void delete()
{
    if (front == -1 && rear == -1)
    {
        printf("Underflow...");
    }
    else
    {
        ele = q[front];
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
        {
            front = front + 1;
        }
        printf("Value %d deleted\n", ele);
    }
}

void display()
{
    if (rear == -1)
    {
        printf("Empty queue");
    }
    else
    {
        printf("The present Queue is:\n");
        for (int i = front; i <= rear; i++)
```

```
    {
        printf("%d ", q[i]);
    }
}}
```

Output:

```
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$gcc program10.c
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$./a.out

Enter the size:5
Operation available are
1.Insertion
2.Deletion
3.Traverse
Enter a choice1
Enter the element to insert:6
The element 6 inserted
The present Q is:
6
Enter a choice1
Enter the element to insert:4
The element 4 inserted
The present Q is:
6 4
Enter a choice1
Enter the element to insert:8
The element 8 inserted
The present Q is:
6 4 8
Enter a choice1
Enter the element to insert:9
The element 9 inserted
The present Q is:
```

6 4 8 9

Enter a choice1

Enter the element to insert:10

The element 10 inserted

The present Q is:

6 4 8 9 10

Enter a choice1

Enter the element to insert:8 3

Overflow...

The present Q is:

6 4 8 9 10

Enter a choice2

Value 6 deleted

The present Q is:

4 8 9 10

Enter a choice2

Value 4 deleted

The present Q is:

8 9 10

Enter a choice3

The present Q is:

8 9 10

Experiment 11 **Date: 12.10.2023**

Circular Queue Operations

Aim:

11. Program to implement circular queue operations using arrays

Algorithm:

void main()

1. Start
2. int ch,value;
3. Enter the choice
4. switch(ch)

```
case 1:
enter the value to add
enqueue(value);
print();
break;

case 2:
dequeue();
print();
break;

case 3:
print();
break;

default:
print "Invalid option..."
```

5. Stop.

void enqueue(int value)

1. Start
2. if (rear == capacity - 1)

```
print “Overflow…!”
```

3. else if (front == -1)

```
front = 0;
rear = (rear + 1) % capacity;
```

```
        queue[rear] = value;
        print “Value enqueued to the circular queue”
```

4. Stop.

void dequeue()

1. Start
2. int variable;
3. if (front == -1 && rear == -1)
   ```
   print "Underflow...!"
   ```
4. else
   ```
   variable = queue[front];
   if (front == rear)
       front = rear = -1;
   else
       front = (front + 1) % capacity;
   print “Value is  dequeued from the circular queue”
   ```
5. Stop.

void dequeue()

1. Start
2. int i;
3. if (front == -1 && rear == -1)
   ```
   print “Underflow…!”
   ```
4. else
   ```
   printf ("\nThe present queue is: \n");
   for (i = front; i != rear; i = (i + 1) % capacity)
   print queue[i]
   ```
5. Stop

Program:

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

int queue[capacity];
int front = -1, rear = -1;

void enqueue(int value)
{
  if (rear == capacity - 1)
```

```
{
```

```
    printf("Overflow...!");
   }
  else
   {
    if (front == -1)
        front = 0;

    rear = (rear + 1) % capacity;
    queue[rear] = value;
    printf ("%d was enqueued to the circular queue\n", value);
   }
}

void dequeue()
{
 int variable;
 if (front == -1 && rear == -1)
   {
    printf("Underflow...!");
   }
 else
   {
    variable = queue[front];
    if (front == rear)
        {
         front = rear = -1;
        }
    else
        {
         front = (front + 1) % capacity;
        }

    printf ("%d was dequeued from the circular queue\n", variable);
   }
}

void print()
{
 int i;
 if (front == -1 && rear == -1)
   {
    printf("Underflow...!");
   }
 else
   {
```

```
        printf ("\nThe present queue is: \n");
        for (i = front; i != rear; i = (i + 1) % capacity)
            {
              printf ("%d ", queue[i]);
            }
        printf ("%d \n\n", queue[i]);

      }
}

void main()
{
int ch,value;
printf("Operations available are: \n1.Enqueue\n2.Dequeue\n3.Display");
do{
printf("\nEnter the choice :");
scanf("%d",&ch);

switch(ch)
{
        case 1:
        printf("Enter the value to add:");
        scanf("%d",&value);
        enqueue(value);
        print();
        break;

        case 2:
        dequeue();
        print();
        break;

        case 3:
        print();
        break;

        default:
        printf("Invalid option...");
}

}while(ch<5);
}
```

Output:

```
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$gcc program11.c
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$./a.out

Operations available are:
1.Enqueue
2.Dequeue
3.Display
Enter the choice :1
Enter the value to add:9
9 was enqueued to the circular queue

The present queue is:
9
Enter the choice :1
Enter the value to add:8
8 was enqueued to the circular queue

The present queue is:
9 8
Enter the choice :1
Enter the value to add:7
7 was enqueued to the circular queue

The present queue is:
9 8 7
Enter the choice :1
Enter the value to add:6
6 was enqueued to the circular queue

The present queue is:
9 8 7 6
Enter the choice :2
9 was dequeued from the circular queue

The present queue is:
8 7 6
Enter the choice :2
8 was dequeued from the circular queue

The present queue is:
7 6
```

```
Enter the choice :2
7 was dequeued from the circular queue

The present queue is:
6
Enter the choice :2
6 was dequeued from the circular queue
Underflow...!
```

Experiment 12 **Date: 19.10.2023**

Singly linked list operations

Aim:

12. To implement the following operations on a singly linked list
    a. Creation,
    b. Insert a new node at front
    c. Insert an element after a particular node
    d. Deletion from beginning
    e. Deletion from the end
    f. Searching
    g. Traversal.

Algorithm:

void main()

1. Start
2. int ch,value,pos;
3. print “Enter an option”
4. switch(ch)

```
case 1:
create();
print "Linked list created with No elements"
break;

case 2:
print "Enter the value:"
insertbegin(value);
display();
break;

case 3:
print "Enter the value:"
print "Enter the position to insert the new node:"
insertposition(value,pos);
display();
break;

case 4:
deletebegin();
```

```
                display();
                break;

                case 5:
                deleteend();
                display(); break;
                default:
                print "Invalid option..."
                break;
```

5. Stop

void create()

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. Stop

void insertbegin(int value)

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. temp->value=value;
5. temp->next=NULL;
6. if(head==NULL)
   head=temp;
7. else
   temp->next=head;
   head=temp;
8. Stop.

void insertposition(int value, int pos)

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. temp->value=value;
5. temp->next=NULL;
6. if(head==NULL)
   head=temp;
7. else
   struct node *prev;
   int count=1;
   ptr=head;
   while(ptr!=NULL)

```
{
prev=ptr;
ptr=ptr->next;
count++;
        if(count==pos)
        {
        temp->next=prev->next;
        prev->next=temp;
        }
}
```

8. Stop.

void deletebegin()

1. Start
2. if(head==NULL)
   print “List empty, No element to delete..."
3. else
   ptr=head;
   head=head->next;
   ptr->next=NULL;
   free(ptr);
4. Stop

void deleteend()

1. Start
2. if(head==NULL)
   print "List empty, No element to delete..."
3. else
   struct node *prev;
   ptr=head;
   while(ptr->next!=NULL)
   prev=ptr;
   ptr=ptr->next;
   if(prev != NULL)
   prev->next = NULL;
   free(ptr);
   else
   free(head);
   head = NULL;
4. Stop

void search()

1. Start
2. Int search_ele,count=1;
3. Enter the element to search.
4. if(head==NULL)
   print "The list is empty..."
5. else
   ptr=head;
   while(ptr != NULL)
   {
   if(ptr->value == search_ele)
   print “Element present”
   break;
   ptr=ptr->next;
   count++;
   }
   If(ptr==NULL)
   Print “Element not present”
6. Stop

void display()

7. Start
8. if(head==NULL)
   print "The list is empty..."
9. else
   ptr=head;
   while(ptr!=NULL)
   print “ptr->value”
   ptr=ptr->next;
10. Stop

Program:

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

struct node{
   int value;
   struct node*next;
};
struct node *head=NULL, *ptr;
```

```
void create()
{
   struct node *temp=NULL;
   temp=malloc(sizeof(struct node));
}

void insertbegin(int value)
{
   struct node *temp=NULL;
   temp=malloc(sizeof(struct node));
   temp->value=value;
   temp->next=NULL;
   if(head==NULL)
   {
      head=temp;
   }
   else
   {
      temp->next=head;
      head=temp;
   }
}

void insertposition(int value,int pos)
{
   struct node *temp=NULL;
   temp=malloc(sizeof(struct node));
   temp->value=value;
   temp->next=NULL;
   if(head==NULL)
   {
      head=temp;
   }
   else
   {
      struct node *prev;
      int count=1;
      ptr=head;
      while(ptr!=NULL)
      {
         prev=ptr;
         ptr=ptr->next;
         count++;
         if(count==pos)
```

```
            {
               temp->next=prev->next;
               prev->next=temp;
            }
         }
      }
   }

   void deletebegin()
   {
      if(head==NULL)
      {
         printf("List empty, No element to delete...");
      }
      else
      {
         ptr=head;
         head=head->next;
         ptr->next=NULL;
         free(ptr);
      }
   }

   void deleteend()
   {
      if (head == NULL)
      {
         printf("List empty, No element to delete...");
      }
      else
      {
         struct node *prev = NULL;
         ptr = head;

         while (ptr->next != NULL)
         {
            prev = ptr;
            ptr = ptr->next;
         }

         if (prev != NULL)
         {
            prev->next = NULL;
            free(ptr);
```

```
            }
            else
            {
                free(head);
                head = NULL;
            }
        }
    }
    void search()
    {
        int search_ele,count=1;
        printf("Enter the element to search:");
        scanf("%d",&search_ele);
        if(head==NULL)
        {
            printf("The list is empty...");
        }
        else
        {
            ptr=head;
            while(ptr != NULL)
            {
                if(ptr->value == search_ele)
                {
                    printf("Element %d present at %d position",search_ele,count);
                    break;
                }
                ptr=ptr->next;
                count++;
            }
            if(ptr==NULL)
            {
                printf("Element not present...");
            }
        }
    }
    void display()
    {
        if(head==NULL)
        {
            printf("The list is empty...");
        }
        else
        {
```

```
        ptr=head;
        while(ptr!=NULL)
        {
           printf("%d\t",ptr->value);
           ptr=ptr->next;
        }
     }
}

void main()
{
   int ch,value,pos;
printf("The Operations available are:\n1)Creation:\n2)Insert at begining:\n3)Insert at a
position:\n4)Delete from begining:\n5)Delete from
end:\n6)Searching:\n7)Traversal:\n8)Exit:");
   do{
   printf("\nEnter an option:");
   scanf("%d",&ch);
   switch(ch)
   {
      case 1:
      {
         create();
         printf("linked list created with No elements");
         break;
      }
      case 2:
      {
         printf("Enter the value:");
         scanf("%d",&value);
         insertbegin(value);
         display();
         break;
      }
      case 3:
      {
         printf("Enter the value:");
         scanf("%d",&value);
         printf("Enter the position to insert the new node:");
         scanf("%d",&pos);
         insertposition(value,pos);
         display();
         break;
      }
```

```
        case 4:
        {
           deletebegin();
           display();
           break;
        }
        case 5:
        {
           deleteend();
           display();
           break;
        }
        case 6:
        {
           search();
           break;
        }
        case 7:
        {
           display();
           break;
        }
        default:
        {
           printf("Invalid option...");
           break;
        }
    }
        }while(ch<8);
}
```

Output:

```
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$gcc program12.c
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$./a.out

The Operations available are:
1)Creation:
2)Insert at begining:
3)Insert at a position:
4)Delete from begining:
5)Delete from end:
6)Searching:
7)Traversal:
```

8)Exit:

```
Enter an option:1
linked list created with No elements

Enter an option:2
Enter the value:10
10

Enter an option:2
Enter the value:20
20      ,10

Enter an option:3
Enter the value:5
Enter the position to insert the new node:3
20      ,10     ,5

Enter an option:6
Enter the element to search:5
Element 5 present at 3 position

Enter an option:6
Enter the element to search:4
Element not present...

Enter an option:4
10      ,5

Enter an option:5
10

Enter an option:7
10

Enter an option:8
Invalid option...
```

Experiment 13 **Date: 20.10.2023**

Doubly linked list operations

Aim:

13. To implement the following operations on a Doubly-linked list.

a. Creation
b. Count the number of nodes
c. Insert a node at first position
d. Insert a node at last
e. Deletion from the first position
f. Deletion from last
g. Searching
h. Traversal.

Algorithm:

void main()

1. Start
2. int ch,item,s_item;
3. Select an option
4. switch(ch)

```
case 1:
create();
break;

case 2:
count();
break;

case 3:
print “Enter the item to be inserted”
insertbegin(item);
break;

case 4:
print “Enter the item to be inserted”
insertlast(item);
break;
```

```
case 5:
deletefirst();
break;

case 6:
deletelast();
break;

case 7:
print "Enter the element to search"
search(s_item);
break;

case 8:
traverse();
break;

default:
print "Invalid option"
```

5. Stop

void create()

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. if(temp == NULL)
   print "Memory not allocated"
5. else
   print "Memory allocated successfully"
6. Stop

void insertbegin(int item)

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. if(temp == NULL)
   print "Insufficient memory"
5. else
   temp->value=item;
   temp->prev=NULL;
   temp->next=NULL;
6. if(head==NULL)
   head=temp;

7. else
   temp->next=head;
   temp->next->prev=temp;
   head=temp;
8. printf “Node inserted”
9. Stop

void insertlast(int item)

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. if(temp == NULL)
   print “Insufficient memory”
5. else
   temp->value=item;
   temp->prev=NULL;
   temp->next=NULL;
6. if(head==NULL)
   head=temp;
7. else
   ptr=head;
   while(ptr->next != NULL)
   ptr=ptr->next;
   ptr->next=temp;
   temp->prev=ptr;
8. printf “Node inserted at last”
9. stop.

void deletefirst ()

1. Start
2. if(head==NULL)
   print “Underflow”
3. else
   ptr=head;
   ptr->next->prev=NULL;
   head=head->next;
   free(ptr);
4. print “Node deleted”
5. Stop.

void deletelast ()

1. Start
2. if(head==NULL)

```
        print “Underflow”
```

3. else
```
        ptr=head;
        while(ptr->next != NULL)
        {
                ptr=ptr->next;
        }
        ptr->prev->next=NULL;
        free(ptr);
        print “Node deleted from the last”
```
4. Stop

void search (int s_item)

1. Start
2. int count=0;
3. if(head==NULL)
```
        printf("Underflow");
```
4. else
```
        ptr=head;
        while(ptr != NULL)
        {
        count++;
        if(ptr->value == s_item)
                print “Item found”
        else
                ptr=ptr->next;
        if(ptr == NULL)
                print “Item present”
        }
```
5. Stop

void count ()

1. Start
2. int count=0;
3. if(head==NULL)
```
        printf("Underflow");
```
4. else
```
        ptr=head;
        while(ptr != NULL)
                ptr=ptr->next;
                count++;
```
5. print “count”
6. Stop

void traverse()

1. Start
2. if(head==NULL)
   print “Underflow”
3. else
   ptr=head;
   while(ptr != NULL)
   print “ptr->value”
   ptr=ptr->next;
4. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int value;
   struct node *next;
   struct node *prev;
};
struct node *head=NULL, *ptr;
void create()
{
   struct node *temp=NULL;
   temp=malloc(sizeof(struct node));
   if(temp==NULL)
   {
      printf("Memory not allocated");
   }
   else
   {
      printf("Memory allocated sucessfully");
   }
}

void insertbegin(int item)
{
   struct node *temp=NULL;
   temp=malloc(sizeof(struct node));
   if(temp==NULL)
```

```
    {
      printf("Insufficient memory");
    }
    else
    {
       temp->value=item;
       temp->prev=NULL;
       temp->next=NULL;
    }
    if(head==NULL)
    {
       head=temp;
    }
    else
    {
       temp->next=head;
       temp->next->prev=temp;
       head=temp;
    }
    printf("node inserted with value = %d",item);
}

void insertlast(int item)
{
    struct node *temp=NULL;
    temp=malloc(sizeof(struct node));
    if(temp==NULL)
    {
      printf("Insufficient memory");
    }
    else
    {
       temp->value=item;
       temp->prev=NULL;
       temp->next=NULL;
    }
    if(head==NULL)
    {
       head=temp;
    }
    else
    {
       ptr=head;
       while(ptr->next != NULL)
```

```
        {
            ptr=ptr->next;
        }
        ptr->next=temp;
        temp->prev=ptr;
    }
    printf("node inserted at LAST with value = %d",item);
}

void deletefirst()
{
        if(head==NULL)
        {
                printf("Underflow");
        }
        else
        {
                ptr=head;
                ptr->next->prev=NULL;
                head=head->next;
                free(ptr);
        }

        printf("node deleted");
}

void deletelast()
{
        if(head==NULL)
        {
                printf("Underflow");
        }
        else
        {
                ptr=head;
                while(ptr->next != NULL)
                {
                        ptr=ptr->next;
                }
                ptr->prev->next=NULL;
                free(ptr);
                printf("Node deleted from last");
        }
}
```

```
void search(int s_item)
{
        int count=0;
        if(head==NULL)
        {
                printf("Underflow");
        }
        else
        {
                ptr=head;
                while(ptr != NULL)
                {
                count++;
                        if(ptr->value == s_item)
                        {
                                printf("item found at position %d",count);
                                break;
                        }
                        else
                        {

                        ptr=ptr->next;
                        }
                }
                if(ptr == NULL)
                {
                printf("Item not present");
                }
        }
}

void count()
{
        int count=0;
        if(head==NULL)
        {
                printf("Undeflow");
        }
        else
        {
                ptr=head;
                while(ptr != NULL)
                {
                        ptr=ptr->next;
                        count++;
```

```
                }
        }
        printf("The total number of node is : %d",count);
}

void traverse()
{
        if(head==NULL)
        {
        printf("Underflow");
        }
        else
        {
        ptr=head;
        while(ptr != NULL)
        {
        printf("%d\t",ptr->value);
        ptr=ptr->next;
        }
        }
}

void main()
{
        int ch,item,s_item;
        printf("The operations available are:\n1.Creation\n2.Count the number of
        nodes\n3.Insertion at begining\n4.Insertion at end\n5.Deletion from
        first\n6.Deletion from last\n7.Searching\n8.Traversal");
        do{
        printf("\nSelect an option:");
        scanf("%d",&ch);

        switch(ch)
        {
        case 1:
        create();
        break;

        case 2:
        count();
        break;

        case 3:
        printf("Enter the item to be inserted:");
```

```
            scanf("%d",&item);
            insertbegin(item);
            break;

            case 4:
            printf("Enter the item to be inserted:");
            scanf("%d",&item);
            insertlast(item);
            break;

            case 5:
            deletefirst();
            break;

            case 6:
            deletelast();
            break;

            case 7:
            printf("Enter the element to search:");
            scanf("%d",&s_item);
            search(s_item);
            break;

            case 8:
            traverse();
            break;

            default:
            printf("Invalid option");
            }
            }while(ch<10);
}
```

Output:

```
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$gcc program13.c
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$./a.out

The operations available are:
1.Creation
2.Count the number of nodes
3.Insertion at begining
4.Insertion at end
```

```
5.Deletion from first
6.Deletion from last
7.Searching
8.Traversal
Select an option:1
Memory allocated sucessfully

Select an option:2
Undeflow The total number of node is : 0

Select an option:3
Enter the item to be inserted:10
node inserted with value = 10

Select an option:3
Enter the item to be inserted:20
node inserted with value = 20

Select an option:8
20      10

Select an option:4
Enter the item to be inserted:5
node inserted at LAST with value = 5

Select an option:8
20      10      5

Select an option:4
Enter the item to be inserted:1
node inserted at LAST with value = 1

Select an option:8
20      10      5       1

Select an option:7
Enter the element to search:5
item found at position 3

Select an option:7
Enter the element to search:99
Item not present

Select an option:5
```

```
node deleted

Select an option:8
10      5       1
Select an option:6
Node deleted from last

Select an option:8
10      5
Select an option:2
The total number of node is : 2
```

Experiment 14 **Date: 27.10.2023**

Stack operations using linked list

Aim:

14. To implement a menu driven program to perform following stack operations using linked list
    a. push
    b. pop
    c. Traversal

Algorithm:

void main()

1. Start
2. int ch,item;
3. print "Enter your choice"
4. switch(ch)

```
case 1:
create();
break;

case 2:
print "Enter the element to be added"
push(item);
print "Currently the elements in the stack are:"
display();
break;

case 3:
pop();
print "Currently the elements in the stack are:"
display(); break;
default:
print "Invalid option"
break;
```

5. Stop.

void create()

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. print “Node created successfully”
5. Stop

void push(int item)

1. Start
2. struct node *temp=NULL;
3. temp=malloc(sizeof(struct node));
4. if(temp==NULL)
   print “Memory not allocated”
5. else
   temp->item=item;
   temp->next=NULL;
6. if(head==NULL)
   head=temp;
7. else
   ptr=head;
   while(ptr->next != NULL)
   ptr=ptr->next;
   ptr->next=temp;
8. print “item pushed to the stack”
9. Stop

void pop()

1. Start
2. struct node *prev;
3. if(head==NULL)
   print “No element to delete”
4. else
   ptr=head;
   while(ptr->next != NULL)
   prev=ptr;
   ptr=ptr->next;

   if (prev != NULL)
   prev->next = NULL;
   print “Element popped”
   free(ptr);

```
else
        print "Element popped"
        free(head);
        head = NULL;
```

5. Stop

void display()

1. Start
2. if(head==NULL)
   print "Stack is empty"
3. else
   ptr=head;
   while(ptr != NULL)
   print ptr->item
   ptr=ptr->next;
4. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int item;
   struct node *next;
};
struct node *head=NULL, *ptr;

void create()
{
   struct node *temp=NULL;
   temp=malloc(sizeof(struct node));
   printf("Node created sucessfully");
}

void push(int item)
{
   struct node *temp=NULL;
   temp=malloc(sizeof(struct node));
   if(temp==NULL)
   {
```

```
        printf("No memory allocated...");
```

```
    }
    else
    {
        temp->item=item;
        temp->next=NULL;
    }
    if(head==NULL)
    {
        head=temp;
    }
    else
    {
        ptr=head;
        while(ptr->next != NULL)
        {
            ptr=ptr->next;
        }
        ptr->next=temp;
    }
    printf("%d pushed to the stack",temp->item);
}

void pop()
{
    struct node *prev = NULL;
    if (head == NULL)
    {
        printf("No element to delete...");
    }
    else
    {
        ptr = head;
        while (ptr->next != NULL)
        {
            prev = ptr;
            ptr = ptr->next;
        }

        if (prev != NULL)
        {
            prev->next = NULL;
            printf("%d popped", ptr->item);
            free(ptr);
        }
```

```
        else
        {
            printf("%d popped", head->item);
            free(head);
            head = NULL;
        }
    }
}
void display()
{
    if(head==NULL)
    {
        printf("Stack is empty");
    }
    else
    {
        ptr=head;
        while(ptr != NULL)
        {
            printf("%d\t",ptr->item);
            ptr=ptr->next;
        }
    }
}

void main()
{
    int ch,item;
    printf("Operations available:\n1.Create\n2.Push\n3.Pop");
    do{
    printf("\nEnter your choice:");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
        {
            create();
            break;
        }
        case 2:
        {
        printf("Enter the element to be added:");
        scanf("%d",&item);
        push(item);
```

```
        printf("\nCurrently the elements in the stack are:");
        display();
        break;
        }
        case 3:
        {
           pop();
           printf("\nCurrently the elements in the stack are:");
           display();
           break;
        }
        default:
        {
           printf("Invalid option");
           break;
        }
    }
}while(ch<4);
}
```

Output:

```
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$gcc program14.c
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$./a.out

Operations available:
1.Create
2.Push
3.Pop
Enter your choice:1
Node created sucessfully
Enter your choice:2
Enter the element to be added:9
9 pushed to the stack
Currently the elements in the stack are:9
Enter your choice:2
Enter the element to be added:8
8 pushed to the stack
Currently the elements in the stack are:9        8
Enter your choice:2
Enter the element to be added:7
7 pushed to the stack
Currently the elements in the stack are:9        8        7
Enter your choice:2
```

```
Enter the element to be added:6
6 pushed to the stack
Currently the elements in the stack are:9 8 7 6
Enter your choice:3
6 popped
Currently the elements in the stack are:9 8 7
Enter your choice:3
7 popped
Currently the elements in the stack are:9 8
Enter your choice:3
8 popped
Currently the elements in the stack are:9
```

Experiment 15 **Date: 27.10.2023**

Queue operations using linked list

Aim:

15. To implement a menu driven program to perform following Queue operations using linked list
    a. enqueue
    b. dequeue
    c. Traversal

Algorithm:

void main()

1. Start
2. int item,ch;
3. print "Enter any option"
4. switch(ch)
   case 1:
   print "Enter the element to be added"
   enqueue(item);
   traversal();
   break;

   case 2:
   dequeue();
   traversal();
   break;

   case 3:
   traversal();
   break;
5. Stop

void enqueue(int item)

1. Start
2. struct node *temp;
3. temp=malloc(sizeof(struct node));
4. if(temp==NULL)
   print "Memory not allocated"

5. else
   temp->value=item;
   temp->next=NULL;
6. if(head==NULL)
   head=temp;
7. else
   ptr=head;
   while(ptr->next != NULL)
   ptr=ptr->next;
   ptr->next=temp;
8. print "Value inserted"

void dequeue()

1. Start
2. if(head==NULL)
   print "Queue is empty"
3. else
   ptr=head;
   head=head->next;
   print "Value deleted"
   free(ptr);
4. Stop

void traversal()

1. Start
2. if(head==NULL)
   print "Queue is empty"
3. else
   ptr=head;
   while(ptr != NULL)
   print "ptr->value"
   ptr=ptr->next;
4. Stop

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int value;
```

```
    struct node *next;
};
struct node *head=NULL, *ptr;
void enqueue(int item)
{
    struct node *temp;
    temp=malloc(sizeof(struct node));
    if(temp==NULL)
    {
        printf("Memory not allocated");
    }
    else
    {
        temp->value=item;
        temp->next=NULL;
    }
    if(head==NULL)
    {
        head=temp;
    }
    else
    {
        ptr=head;
        while(ptr->next != NULL)
        {
            ptr=ptr->next;
        }
        ptr->next=temp;
    }
    printf("%d inserted",item);
}
```

```
void dequeue()
{
   if(head==NULL)
   {
      printf("\nQueue is empty");
   }
   else
   {
      ptr=head;
      head=head->next;
      printf("%d is deleted",ptr->value);
      free(ptr);
   }
}
void traversal()
{
   if(head==NULL)
   {
      printf("Queue is empty");
   }
   else
   {
      ptr=head;
      printf("\nThe present Queue is:");
      while(ptr != NULL)
      {
         printf("%d \t",ptr->value);
         ptr=ptr->next;
      }
   }
}
```

```
void main()
{
   int item,ch;
   printf("The operations available:\n1.Enqueue:\n2.Dequeue:\n3.Traversal");
   do{
      printf("\nEnter any option:");
      scanf("%d",&ch);
      switch(ch)
      {
         case 1:
         {
            printf("Enter the element to be added:");
            scanf("%d",&item);
            enqueue(item);
            traversal();
            break;
         }
         case 2:
         {
         dequeue();
         traversal();
         break;
         }
         case 3:
         {
            traversal();
            break;
         }
      }
   }while(ch<4);
}
```

Output:

```
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$gcc program15.c
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$./a.out

The operations available:
1.Enqueue:
2.Dequeue:
3.Traversal
Enter any option:1
Enter the element to be added:9
9 inserted
The present Queue is:9
Enter any option:1
Enter the element to be added:8
8 inserted
The present Queue is:9  8
Enter any option:1
Enter the element to be added:7
7 inserted
The present Queue is:9  8  7
Enter any option:1
Enter the element to be added:6
6 inserted
The present Queue is:9  8  7  6
Enter any option:2
9 is deleted
The present Queue is:8  7  6
Enter any option:2
8 is deleted
The present Queue is:7  6
```

```
Enter any option:2

7 is deleted

The present Queue is:6

Enter any option:2

6 is deletedQueue is empty
```

Experiment 16 **Date: 02.11.2023**

Binary Search Tree (BST) operations

Aim:

16. Menu Driven program to implement Binary Search Tree (BST) Operations.
    a. Insertion of node
    b. Deletion of a node
    c. In-order Traversal
    d. Pre-order Traversal
    e. Post-order Traversal

Algorithm:

int main()

1. Start
2. int choice, value;
3. print “Enter your choice”
4. switch (choice)

```
case 1:
print “Enter the value to insert:”
root = insert(root, value);
break;

case 2:
print “Enter the value to delete:”
root = delete(root, value);
break;

case 3:
inorder(root);
break;

case 4:
preorder(root);
break;

case 5:
postorder(root);
break;

default:
print “Invalid choice”
```

5. return 0;
6. Stop

struct node* new_node(int x)

1. Start
2. struct node *temp;
3. temp = malloc(sizeof(struct node));
4. temp->data = x;
5. temp->left_child = NULL;
6. temp->right_child = NULL;
7. return temp;
8. Stop

struct node* insert(struct node* root, int x)

1. Start
2. if (root == NULL)
   return new_node(x);
3. else if (x > root->data)
   root->right_child = insert(root->right_child, x);
4. else
   root->left_child = insert(root->left_child, x);
5. return root;
6. Stop

struct node* find_minimum(struct node* root)

1. Start
2. if (root == NULL)
   return NULL;
3. else if (root->left_child != NULL)
   return find_minimum(root->left_child);
4. return root;
5. Stop.

struct node* delete(struct node* root, int x)

1. Start
2. if (root == NULL)
   return NULL;
3. if (x > root->data)
   root->right_child = delete(root->right_child, x);
4. else if (x < root->data)

root->left_child = delete(root->left_child, x);

5. else if (root->left_child == NULL && root->right_child == NULL)
   free(root);
   return NULL;
6. else if (root->left_child == NULL || root->right_child == NULL)
   struct node* temp;
   if (root->left_child == NULL)
   temp = root->right_child;
   else
   temp = root->left_child;
   free(root);
   return temp;
7. else
   struct node* temp = find_minimum(root->right_child);
   root->data = temp->data;
   root->right_child = delete(root->right_child, temp->data);
8. return root;
9. Stop

void inorder(struct node* root)

1. Start
2. if (root != NULL)
   inorder(root->left_child);
   print "root->data"
   inorder(root->right_child);
3. Stop

void preorder(struct node* root)

1. Start
2. if (root != NULL)
   print "root->data"
   preorder(root->left_child);
   preorder(root->right_child);
3. Stop

void postorder(struct node* root)

1. Start
2. if (root != NULL)
   postorder(root->left_child);
   postorder(root->right_child);
   print " root->data"
3. Stop

Program:

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

struct node
   {
    int data;
    struct node *right_child;
    struct node *left_child;
   };

struct node* new_node(int x)
   {
    struct node *temp;
    temp = malloc(sizeof(struct node));
    temp->data = x;
    temp->left_child = NULL;
    temp->right_child = NULL;

    return temp;
   }

struct node* insert(struct node* root, int x)
   {
      if (root == NULL)
         return new_node(x);
      else if (x > root->data)
         root->right_child = insert(root->right_child, x);
      else
         root->left_child = insert(root->left_child, x);
      return root;
   }

struct node* find_minimum(struct node* root)
   {
      if (root == NULL)
         return NULL;
      else if (root->left_child != NULL)
         return find_minimum(root->left_child);
      return root;
   }
```

```
struct node* delete(struct node* root, int x)
{
   if (root == NULL)
      return NULL;
   if (x > root->data)
      root->right_child = delete(root->right_child, x);
   else if (x < root->data)
      root->left_child = delete(root->left_child, x);
   else
   {
      if (root->left_child == NULL && root->right_child == NULL)
      {
         free(root);
         return NULL;
      }
      else if (root->left_child == NULL || root->right_child == NULL)
      {
         struct node* temp;
         if (root->left_child == NULL)
            temp = root->right_child;
         else
            temp = root->left_child;
         free(root);
         return temp;
      }
      else
      {
         struct node* temp = find_minimum(root->right_child);
         root->data = temp->data;
         root->right_child = delete(root->right_child, temp->data);
      }
   }
   return root;
}

void inorder(struct node* root)
{
   if (root != NULL)
   {
      inorder(root->left_child);
      printf(" %d ", root->data);
      inorder(root->right_child);
   }
```

```
}

void preorder(struct node* root)
{
   if (root != NULL)
   {
      printf(" %d ", root->data);
      preorder(root->left_child);
      preorder(root->right_child);
   }
}

void postorder(struct node* root)
{
   if (root != NULL)
   {
      postorder(root->left_child);
      postorder(root->right_child);
      printf(" %d ", root->data);
   }
}

int main()
{
   struct node* root = NULL;
   int choice, value;

   printf("Operations available are:\n1. Inserting a new element\n2. Deletion of a node\n3. Display (In-Order)\n4. Display (Pre-Order)\n5. Display (Post-Order)");
   do{
   printf("\nEnter your choice: ");
   scanf("%d", &choice);

   switch (choice)
   {
    case 1:
      printf("Enter the value to insert: ");
      scanf("%d", &value);
      root = insert(root, value);
      break;

    case 2:
      printf("Enter the value to delete: ");
      scanf("%d", &value);
```

```
        root = delete(root, value);
        break;

      case 3:
        printf("Inorder Traversal: ");
        inorder(root);
        printf("\n");
        break;

      case 4:
        printf("Pre-Order Traversal: ");
        preorder(root);
        printf("\n");
        break;

     case 5:
        printf("Post-Order Traversal: ");
        postorder(root);
        printf("\n");
        break;

      default:
        printf("Invalid choice\n");
    }
  } while (choice<6);

  return 0;
}
```

Output:

```
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$gcc program16.c
mits@mits-Lenovo-S510:~/Desktop/S1 MCA$./a.out

Operations available are:
1. Inserting a new element
2. Deletion of a node
3. Display (In-Order)
4. Display (Pre-Order)
5. Display (Post-Order)
Enter your choice: 1
Enter the value to insert: 100

Enter your choice: 1
```

```
Enter the value to insert: 50

Enter your choice: 1
Enter the value to insert: 80

Enter your choice: 1
Enter the value to insert: 40

Enter your choice: 1
Enter the value to insert: 120

Enter your choice: 1
Enter the value to insert: 110

Enter your choice: 1
Enter the value to insert: 150

Enter your choice: 3
Inorder Traversal:  40  50  80  100  110  120  150

Enter your choice: 2
Enter the value to delete: 150

Enter your choice: 2
Enter the value to delete: 50

Enter your choice: 3
Inorder Traversal:  40  80  100  110  120

Enter your choice: 4
Pre-Order Traversal:  100  80  40  120  110

Enter your choice: 5
Post-Order Traversal:  40  80  110  120  100

Enter your choice:
```