**Experiment 1**                                    **Date: 21.09.2023**

**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>
   void main(){
      int a,b;
      printf("Enter 2 numbers : ");
      scanf("%d %d",&a,&b);
      printf("Sum : %d",a+b);
   }
   ```

   **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 2 numbers : 10 20
Sum : 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 named sum_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 in the 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 -lm

Here, 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 the resulting executable.

**Option: -O**

To optimize for code size and execution time.

**<u>Output</u>**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -O3 -o my_pgm sum.c

This compiles sum.c with a high level of optimization.

**Option: -pg**

To enable code profiling.

**<u>Output</u>**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -pg -o my_pgm sum.c

This compiles source.c with profiling support, allowing you to use profilers like gprof.

**Option: -save-temps**

To save temporary files generated during program execution.

**<u>Output</u>**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -save-temps -o my_pgm sum.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**

<div align="center">

**Familiarisation with GDB**

</div>

## 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**

   ```c
   #include<stdio.h>
   void main(){
      int a,b;
      printf("Enter 2 numbers : ");
      scanf("%d %d",&a,&b);
      printf("Product : %d",a*b);
   }
   ```

   **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/Poojas1MCA/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.

## <u>Output</u>

(gdb) print a
(gdb) a 10

**Command: display**

Displays the current values of the specified variable after every step.

## <u>Output</u>

(gdb) display a
1: a=10

**Experiment 3**                                              **Date: 29.09.2023**

## Familiarisation with gprof

**Aim:**

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

### Program

```
#include<stdio.h>
int sum(int x, int y){
    return x+y;
}
void main(){
    int a,b;
    printf("Enter 2 numbers : ");
    scanf("%d %d",&a,&b);
    printf("Sum : %d",sum(a,b));
}
```

### Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc sum.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc ./a.out sum.c
Enter 2 numbers : 10 20
Sum : 30

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc -o sum.out -pg sum.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./sum.out
Enter 2 numbers : 10 20
Sum : 30

mits@mits:~/Desktop/S1MCA/ADS_lab$ gprof ./sum.out gmon.out > pgm3.txt

**pgm3.txt**

Flat profile:

Each sample counts as 0.01 seconds.
 no time accumulated

```
 %  cumulative self       self   total
 time  seconds  seconds  calls Ts/call Ts/call name
 0.00    0.00    0.00     1   0.00   0.00  sum
```

**Experiment 4**                                    **Date: 29.09.2023**

### Different types of functions

**Aim:**

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

   **Algorithm:**

   **main()**

   1. Start
   2. Declare ch,a,b.
   3. Display choices.
   4. Read option ch.
      a. if ch==1 call sum1().
      b. if ch==2 input a and b and call sum2().
      c. if ch==3 print sum3().
      d. if ch==3 input a and b and print sum4().
   5. Repeat steps 3 while ch>0&&ch<4.
   6. Stop.

   **void sum1()**

   1. Start
   2. Declare a and b.
   3. Read a and b.
   4. Print a+b.
   5. Exit.

   **void sum2(int a, int b)**

   1. Start
   2. Print a+b.
   3. Exit.

   **int sum3()**

   1. Start
   2. Declare a and b.
   3. Read a and b.
   4. Return a+b.
   5. Exit.

**int sum4(int a, int b)**

1. Start
2. Return a+b
3. Exit.

## Program

```c
#include<stdio.h>
void sum1(){
   int a,b;
   printf("Enter 2 numbers : ");
   scanf("%d %d",&a,&b);
   printf("Sum : %d",a+b);
}
void sum2(int a, int b){
   printf("Sum : %d",a+b);
}
int sum3(){
   int a,b;
   printf("Enter 2 numbers : ");
   scanf("%d %d",&a,&b);
   return a+b;
}
int sum4(int a, int b){
   return a+b;
}
void main(){
   int ch,a,b;
   do{
      printf("1. Function without return type and arguments\n2. Function without
return type and with arguments\n3. Function with return type and without
arguments\n4. Function with return type and arguments\n5. Exit\nEnter your
choice(1-4): ");
      scanf("%d", &ch);
      switch(ch){
         case 1: sum1();
               break;
         case 2: printf("Enter 2 numbers : ");
               scanf("%d %d",&a,&b);
               sum2(a,b);
               break;
```

```
        case 3: printf("Sum : %d",sum3());
                break;
        case 4: printf("Enter 2 numbers : ");
                scanf("%d %d",&a,&b);
                printf("Sum : %d",sum4(a,b));
                break;
    }
  }while(ch>0&&ch<4);
}
```

## **Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM4.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM4.c


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 2 numbers : 10 20
Sum : 30


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 2 numbers : 25 25
Sum : 50


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 2 numbers : 100 100
Sum : 200

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 2 numbers : 250 250
Sum : 500

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

mits@mits:~/Desktop/S1MCA/ADS_lab$

## Experiment 5                                    Date: 06.10.2023

### Array Operations

### Aim:

5. To implement a menu driven program to perform following array operations
   a. Insert an element to a particular location.
   b. Delete an element from a particular location.
   c. Traverse

### Algorithm:

**main()**

1. Start
2. Declare ch,a[50],p,x and n.
3. Display choices.
4. Read option ch.
   a. if ch==1 call insert().
   b. if ch==2 call del().
   c. if ch==3 call display()
5. Repeat steps 3 while ch>0&&ch<4.
6. Stop.

**void insert()**

1. Start
2. Read the element and it's position.
3. Check if p>n if true print Invalid Position.
4. else n++

   ```
   for(int i=n-1;i>=p;i--){
           a[i]=a[i-1];
   }
   a[p-1]=x;
   ```
5. Exit

**void del()**

1. Start
2. Read the position of the element to be deleted.
3. Check if p>n if true print Invalid Position.
4. else n–

   print a[p-1] is deleted

   for(int i=p-1;i<=n-1;i++){
   a[i]=a[i+1];
5. Exit

**void display()**

6. Start
7. for(int i=0;i<n;i++){
           print a[i]
   }
8. Exit

**Program**

```
#include<stdio.h>
int a[50],p,x,n;

void insert(){
   printf("Enter the element and it's position : ");
   scanf("%d %d",&x,&p);
   if(p>n){
      printf("Invalid Position");
   }
   else{
      n++;
      for(int i=n-1;i>=p;i--){
         a[i]=a[i-1];
      }
      a[p-1]=x;
   }
}
void del(){
   printf("Enter position of the element to be deleted : ");
   scanf("%d",&p);
   if(p>n){
```

```c
        printf("Invalid Position");
      }
      else{
        n--;
        printf("%d is deleted",a[p-1]);
        for(int i=p-1;i<=n-1;i++){
          a[i]=a[i+1];
        }
      }
    }
    void display(){
      printf("Array : ");
      for(int i=0;i<n;i++){
        printf("%d \t",a[i]);
      }
    }
    void main(){
      printf("Enter the size of the array : ");
      scanf("%d",&n);
      printf("Enter array : ");
      for(int i=0;i<n;i++){
        scanf("%d",&a[i]);
      }
      int ch;
      do{
        printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice(1-4) : ");
        scanf("%d",&ch);
        switch(ch){
          case 1 :  insert();
                  break;
          case 2 :  del();
                  break;
          case 3 :  display();
                  break;
        }
      }while(ch>0&&ch<4);
    }
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM5.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM5.c

Enter the size of the array : 5
Enter array : 10 20 30 40 50

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element and it's position : 25 3

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 3
Array : 10    20    25    30    40    50

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
Enter position of the element to be deleted : 3
25 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 3
Array : 10    20    30    40    50

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 6**                                    **Date: 06.10.2023**

**Array Sorting**

**Aim:**

6.  Program to sort an integer array

    **Algorithm:**

    **main()**

    1. Start
    2. Declare a[50], and n.
    3. Read n and input n elements to a.
    4. Print a before sorting by calling display().
    5. Call sort().
    6. Print a after sorting by calling display().
    7. Stop.

    **void sort()**

    1. Start
    2. for(int i=0;i<n-1;i++){
           for(int j=0;j<n-i-1;j++){
             if(a[j]>a[j+1]){
               int t=a[j];
               a[j]=a[j+1];
               a[j+1]=t;
             }
           }
        }
    3. Exit

    **void display()**

    1. Start
    2. for(int i=0;i<n;i++){
               print a[i]
        }
    3. Exit

**<u>Program</u>**

```c
#include<stdio.h>
int a[50],n;

void sort(){
for(int i=0;i<n-1;i++){
   for(int j=0;j<n-i-1;j++){
    if(a[j]>a[j+1]){
      int t=a[j];
      a[j]=a[j+1];
      a[j+1]=t;
     }
   }
  }
}

void display(){
   printf("Array : ");
   for(int i=0;i<n;i++){
      printf("%d \t",a[i]);
   }
}

void main(){
   printf("Enter the size of the array : ");
   scanf("%d",&n);
   printf("Enter array : ");
   for(int i=0;i<n;i++){
      scanf("%d",&a[i]);
   }
   printf("Array before sorting : ");
   display();
   sort();
   printf("\nArray after sorting : ");
   display();
}
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM6.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM6.c

Enter the size of the array : 5
Enter array : 50 30 10 20 40

Array before sorting : Array :    50      30      10      20      40
Array after sorting : Array :     10      20      30      40      50

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 7**                                              **Date: 06.10.2023**

## Searching Operations

**Aim:**

7.  Program to implement linear search and binary search.

    **Algorithm:**

    **main()**

    1.  Start
    2.  Declare a[50],n,ch,x and p.
    3.  Read n and input n elements to a.
    4.  Read x.
    5.  Display choices
    6.  Read ch
        a.  if ch==1 call p=l_search(x)
        b.  if ch==2 call p=b_search(x)
    7.  if(p==-1){
            print element not found
        }
        else{
            print element found at position p+1
        }
    8.  Repeat steps 5 to 7 while ch>0&&ch<3.
    9.  Stop.

    **int l_search(int x)**

    1.  Start
    2.  for(int i=0;i<n;i++){
            if(x==a[i]){
                return i;
            }
        }
    3.  return -1;
    4.  Exit

    **int b_search(int x)**

    1.  Start
    2.  Declare b=0,l=n-1 and m
    3.  while b<=l
            m=(b+l)/2;
            if(x==a[m])

```
                        return m;
                else if(x>a[m])
                        b=m+1;
                else
                        l=m-1;
    4. return -1;
    5. Exit
```

**Program**

```c
#include<stdio.h>
int a[50],n;
int l_search(int x){
   for(int i=0;i<n;i++){
     if(x==a[i]){
        return i;
     }
   }
   return -1;
}

int b_search(int x){
   int b=0,l=n-1,m;
   while(b<=l){
     m=(b+l)/2;
     if(x==a[m])
        return m;
     else if(x>a[m])
        b=m+1;
     else
        l=m-1;
   }
   return -1;
}

void main(){
   int ch,x,p;
   printf("Enter the size of the array : ");
   scanf("%d",&n);
   printf("Enter array : ");
   for(int i=0;i<n;i++){
     scanf("%d",&a[i]);
   }
```

```
    printf("Enter the element to be searched : ");
    scanf("%d",&x);
    printf("Select the searching method : ");

    do{
        printf("1. Linear Search\n2. Binary Search\n3. Exit\nEnter your   choice(1-
3) : ");
        scanf("%d",&ch);
        switch(ch){
            case 1  :  p=l_search(x);

            case 2  :  p=b_search(x);

                    if(p==-1){
                    printf("Element not found");
                    }
                    else{
                    printf("Element found at position %d",p+1);
                    }
        }
    }while(ch>0&&ch<3);
}
```

## Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM7.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM7.c

Enter the size of the array : 5
Enter array : 10 20 30 40 50
Enter the element to be searched : 30

Select the searching method :
1. Linear Search
2. Binary Search
3. Exit
Enter your choice(1-3) : 1
Element found at position 3

1. Linear Search
2. Binary Search
3. Exit
Enter your choice(1-3) : 2
Element found at position 3

1. Linear Search
2. Binary Search
3. Exit
Enter your choice(1-3) : 3

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 8**                                             **Date: 08.10.2023**

## Matrix Operations

**Aim:**

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

**Algorithm:**

**main()**

    **1.**

**void add(int a[50][50], int b[50][50], int n)**

1. 1Start
2. Declare c[50][50];
3. for(int i=0;i<n;i++){
       for(int j=0;j<n;j++){
         c[i][j]=a[i][j]+b[i][j];
       }
     }
4. Call display(c,n) to print the resultant matrix
5. Exit

**void sub(int a[50][50], int b[50][50], int n)**

1. Start
2. Declare c[50][50];
3. for(int i=0;i<n;i++){
       for(int j=0;j<n;j++){
         c[i][j]=a[i][j]-b[i][j];
       }
     }
4. Call display(c,n) to print the resultant matrix
5. Exit

**void mul(int a[50][50], int b[50][50], int n)**

1. Start
2. Declare c[50][50];
3. for(int i=0;i<n;i++){
       for(int j=0;j<n;j++){
         c[i][j]=0;
         for(int k=0;k<n;k++){

```
                    c[i][j]+=a[i][k]*b[k][j];
                  }
                }
              }
```
4. Call display(c,n) to print the resultant matrix
5. Exit

**void display(int a[50][50], int n)**

1. Start
2. for(int i=0;i<n;i++){
   ```
           for(int j=0;j<n;j++){
              printf("%d \t",a[i][j]);
           }
           printf("\n");
        }
   ```
3. Exit

**void input(int a[50][50], int n)**

1. Start
2. for(int i=0;i<n;i++){
   ```
           for(int j=0;j<n;j++){
              scanf("%d",&a[i][j]);
           }
        }
   ```
3. Exit

## Program

```c
#include<stdio.h>
int a[50][50],b[50][50],m,n;

void display(int a[50][50], int n){
   for(int i=0;i<n;i++){
     for(int j=0;j<n;j++){
        printf("%d \t",a[i][j]);
     }
     printf("\n");
   }
}

void input(int a[50][50], int n){
   for(int i=0;i<n;i++){
     for(int j=0;j<n;j++){
```

```c
            scanf("%d",&a[i][j]);


        }
    }
}


void add(int a[50][50], int b[50][50], int n){
    int c[50][50];
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            c[i][j]=a[i][j]+b[i][j];
        }
    }
    printf("Resultant Matrix : \n");
    display(c,n);
}


void sub(int a[50][50], int b[50][50], int n){
    int c[50][50];
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            c[i][j]=a[i][j]-b[i][j];
        }
    }
    printf("Resultant Matrix : \n");

    display(c,n);
}


void mul(int a[50][50], int b[50][50], int n){
    int c[50][50];
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            c[i][j]=0;
            for(int k=0;k<n;k++){
                c[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
    printf("Resultant Matrix : \n");
    display(c,n);
}
```

```
void main(){
    int ch;

    printf("Enter the size of the matrix 1 : ");
    scanf("%d",&m);
    printf("Enter matrix 1 : ");
    input(a,m);
    printf("Enter the size of the matrix 2 : ");
    scanf("%d",&n);
    printf("Enter matrix 2 : ");
    input(b,n);

    do{
        printf("1. Add\n2. Subtract\n3. Multiply\n4. Exit\nEnter your choice(1-4) : ");
        scanf("%d",&ch);
        switch(ch){
            case 1 :  add(a,b,n);
                    break;
            case 2 :  sub(a,b,n);
                    break;
            case 3 :  mul(a,b,n);
                    break;
        }
    }while(ch>0&&ch<4);
}
```

## Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM8.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM8.c

Enter the size of the matrix 1 : 2
Enter matrix 1 : 1 2 3 4
Matrix 1 :
1       2
3       4

Enter the size of the matrix 2 : 2
Enter matrix 2 : 5 6 7 8
Matrix 2 :
5       6
7       8

1. Add
2. Subtract
3. Multiply
4. Exit
Enter your choice(1-4) : 1
Resultant Matrix :
6       8
10      12

1. Add
2. Subtract
3. Multiply
4. Exit
Enter your choice(1-4) : 2
Resultant Matrix :
-4      -4
-4      -4

1. Add
2. Subtract
3. Multiply
4. Exit
Enter your choice(1-4) : 3
Resultant Matrix :
19      22
43      50

1. Add
2. Subtract
3. Multiply
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 9**                                               **Date: 12.10.2023**

## Stack Operations

## Aim:

9.  Program to implement stack operations using arrays.

### Algorithm:

**main()**

1.  Start
2.  Declare a[50],n=5, ch and top=-1
3.  Display choices
4.  Read ch
    a.  if ch==1 call push()
    b.  if ch==2 call pop()
    c.  if ch==3 call display()
5.  Stop

**void push()**

1.  Start
2.  if top==n-1 print stack overflow
3.  else

    top++
    read a[top]
4.  Exit

**void pop()**

1.  Start
2.  if top==-1 print stack underflow
3.  else

    print a[top] is deleted
    top–
4.  Exit

**void display()**

1.  Start
2.  if top==-1 print stack underflow
3.  else

    for(int i=top;i>=0;i--)
        print a[i]
4.  Exit

**Program**

```c
#include<stdio.h>
#define n 5
int a[50], top=-1;

void push(){
   if(top==n-1){
      printf("Stack Overflow");
   }
   else{
      top++;
      printf("Enter the element to be inserted : ");
      scanf("%d",&a[top]);
   }
}

void pop(){
   if(top==-1){
      printf("Stack Underflow");
   }
   else{
      printf("%d is deleted",a[top]);
      top--;
   }
}
void display(){
   if(top==-1){
      printf("Stack Underflow");
   }
   else{
      printf("Stack : ");
      for(int i=top;i>=0;i--){
         printf("%d \t",a[i]);
      }
   }
}
void main(){
 int ch;
   do{
      printf("1. Push\n2. Pop\n3. Display\n4. Exit\nEnter your choice(1-4) : ");
      scanf("%d",&ch);
```

```
    switch(ch){

        case 1  :  push();
                   break;
        case 2  :  pop();
                   break;
        case 3  :  display();
                   break;
    }
  }while(ch>0&&ch<4);
}
```

## Output

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM9.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM9.c

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 20

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 30

1. Push
2. Pop
3. Display
4. Exit

Enter your choice(1-4) : 1
Enter the element to be inserted : 40

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 50

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Stack Overflow

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 3
Stack : 50    40    30    20    10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
50 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
40 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
30 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
20 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
10 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
Stack Underflow

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 3
Stack Underflow

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 10**                                                           **Date: 12.10.2023**

## Queue Operations

**Aim:**

10. Program to implement queue operations using arrays.

### Algorithm:

**main()**

1. Start
2. Declare a[50],n=5, ch, f=-1 and r=-1
3. Display choices
4. Read ch
     a. if ch==1 call enqueue()
     b. if ch==2 call dequeue()
     c. if ch==3 call display()
5. Stop

**void enqueue()**

1. Start
2. if r==n-1 print queue is full
3. else

       if f==r

             set f=r=0

       else

             r++

       read a[r]
4. Exit

**void dequeue()**

1. Start
2. if f==-1 print queue is empty
3. else

       print a[f] is deleted

       if(f==r)

             set f=r=-1

       else

             f++
4. Exit

**void display()**

1. Start
2. if r==-1 print queue underflow
3. else

        for(int i=f;i<=r;i++)
                print a[i]
4. Exit.

## **Program**

```c
#include<stdio.h>
#define n 5
int a[50],f=-1,r=-1;
void enqueue(){
   if(r==n-1){
      printf("Queue is full");
   }
   else{
      if(f==-1)
         f=r=0;
      else
         r++;
      printf("Enter the element to be inserted : ");
      scanf("%d",&a[r]);
   }
}

void dequeue(){
   if(f==-1){
      printf("Queue is empty");
   }
   else{
      printf("%d is deleted",a[f]);
      if(f==r)
         f=r=-1;
      else
         f++;
   }
}

void display(){
   if(f==-1){
```

```c
        printf("Queue is empty");
      }
    else{
       printf("Queue : ");
       for(int i=f;i<=r;i++){
          printf("%d \t",a[i]);
       }
    }
}

void main(){
 int ch;
   do{
      printf("1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice(1-4) : ");
      scanf("%d",&ch);

      switch(ch){
         case 1 :  enqueue();
                 break;
         case 2 :  dequeue();
                 break;
         case 3 :  display();
                 break;
      }
   }while(ch>0&&ch<4);
}
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM10.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM10.c

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 30

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 40

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 50

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Queue is full

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 3
Queue : 10    20    30    40    50

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
10 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
20 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
30 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
40 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
50 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
Queue is empty

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 11**                                               **Date: 12.10.2023**

## Circular Queue Operations

## Aim:

11. Program to implement circular queue using array.

### Algorithm:

**main()**

1. Start
2. Declare ch,a[50],f=-1,r=-1 and n.
3. Display choices.
4. Read option ch.
   a. if ch==1 call enqueue().
   b. if ch==2 call dequeue().
   c. if ch==3 call display()
5. Repeat steps 3 while ch>0&&ch<4.
6. Stop.

**void enqueue()**

1. Start
2. if (r+1)%n==f print queue is full
3. else

       if f==-1

           set f=r=0

       else

           r=(r+1)%n;

       read a[r]

4. Exit

**void dequeue()**

1. Start
2. if f==-1 print queue is empty
3. else

       print a[f] is deleted

       if(f==r)

           set f=r=-1

       else

           f=(f+1)%n

4. Exit

**void display()**

1. Start
2. if f==-1 print queue underflow
3. else

```
        for(i=f;i!=r;i=(i+1)%n){
                printf("%d \t",a[i]);
        }
        printf("%d \t",a[i]);
```

4. Exit.

## Program

```c
#include<stdio.h>
#define n 5
int a[50],f=-1,r=-1;
void enqueue(){
   if((r+1)%n==f){
      printf("Queue is full");
   }
   else{
      if(f==-1)
         f=r=0;
      else
         r=(r+1)%n;
      printf("Enter the element to be inserted : ");
      scanf("%d",&a[r]);
   }
}

void dequeue(){
   if(f==-1){
      printf("Queue is empty");
   }
   else{
      printf("%d is deleted",a[f]);
      if(f==r)
         f=r=-1;
      else
         f=(f+1)%n;
   }
}
```

```c
    void display(){
      int i;
      if(f==-1){
        printf("Queue is empty");
      }
      else{
        printf("Queue : ");
        for(i=f;i!=r;i=(i+1)%n){
          printf("%d \t",a[i]);
        }
        printf("%d \t",a[i]);
      }
    }

    void main(){
     int ch;
      do{
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\nEnter your choice(1-4)
: ");
        scanf("%d",&ch);

        switch(ch){
          case 1 :  enqueue();
                  break;
          case 2 :  dequeue();
                  break;
          case 3 :  display();
                  break;
        }
      }while(ch>0&&ch<4);
    }
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM11.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM11.c

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 10

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 20

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 30

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 40

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 50

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Queue is full

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 3
Queue : 10   20    30    40    50

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
10 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 2
20 is deleted

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 60

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the element to be inserted : 70

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 3
Queue : 30   40      50      60      70

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$

## 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
    d. Deletion from beginning
    e. Deletion from the end
    f. Searching
    g. Traversal.

### Algorithm:

**main()**

1. Start
2. struct node{
        int data;
        struct node *next;
   }*head, *ptr, *temp;
3. Display choices.
4. Read option ch.
       a. if ch==1 call ins_beg().
       b. if ch==2 call ins_spec().
       c. if ch==3 call del_beg()
       d. if ch==4 call del_end()
       e. if ch==5 call search()
       f. if ch==6 call display()
5. Repeat step 3 while ch>0&&ch<7.
6. Stop.

**void ins_beg()**

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
          if head==NULL
                ptr->next=NULL;
                head=ptr
          else
                ptr->next=head;

head=ptr
4. Exit


**void ins_spec()**

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. set temp=head
5. for(int i=1;i<p;i++){
           temp=temp->next;
           if(temp==NULL){
                   printf("Invalid Position");
                   break;
           }
    }
    ptr->next=temp->next;
    temp->next=ptr;
6. Exit

**void del_beg()**

1. Start
2. if head==NULL print List Empty
3. else
           print head->data is deleted
           if head->next==NULL
                   free(head);
                   head=NULL;
           else
                   ptr=head;
                   head=ptr->next;
                   free(ptr);
4. Exit.

**void del_end()**

1. Start
2. if head==NULL print List Empty
3. else

```
if head->next==NULL
        print head->data is deleted
        free(head);
        head=NULL;
else
        ptr=head;
        while(ptr->next!=NULL){
                temp=ptr;
                ptr=ptr->next;
        }
printf("%d is deleted",ptr->data);
temp->next=NULL;
free(ptr);
```

4. Exit.

**void display()**

1. Start
2. if head==NULL print List Empty
3. else

```
printf("Linked List : ");
while(ptr!=NULL){
        printf("%d\t",ptr->data);
        ptr=ptr->next;
}
```

4. Exit.

**void search()**

1. Start
2. Declare x,i=1,f=0
3. if head==NULL print List Empty
4. else

   read x
   for(ptr=head; ptr!=NULL; ptr=ptr->next){
       if(ptr->data==x){
               print element found at node i
               set f=1
       }
       i++
   }
   if f ==0 print Element not found
5. Exit.

## Program

```c
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct node *next;
}*head, *ptr, *temp;

void ins_beg(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   if(head==NULL){
     ptr->next=NULL;
     head=ptr;
   }
   else{
     ptr->next=head;
     head=ptr;
   }
}
```

```c
void ins_spec(){
    int p;
    ptr =  malloc(sizeof(struct node));
    printf("Enter the item and it's position : ");
    scanf("%d %d",&ptr->data,&p);
    temp=head;
    for(int i=1;i<p;i++){
        temp=temp->next;
        if(temp==NULL){
            printf("Invalid Position");
            break;
        }
    }
    ptr->next=temp->next;
    temp->next=ptr;
}

void del_beg(){
    if(head==NULL){
        printf("List Empty");
    }
    else{
        printf("%d is deleted",head->data);
        if(head->next==NULL){
            free(head);
            head=NULL;
        }
        else{
            ptr=head;
            head=ptr->next;
            free(ptr);
        }
    }
}

void del_end(){
    if(head==NULL){
        printf("List Empty");
    }
    else{
        if(head->next==NULL){
            printf("%d is deleted",head->data);
```

```
                free(head);
                head=NULL;
            }

            else{
                ptr=head;
                while(ptr->next!=NULL){
                    temp=ptr;
                    ptr=ptr->next;
                }
                printf("%d is deleted",ptr->data);
                temp->next=NULL;
                free(ptr);
            }
        }
    }

    void display(){
        if(head==NULL){
            printf("List Empty");
        }
        else{
            ptr=head;
            printf("Linked List : ");
            while(ptr!=NULL){
                printf("%d\t",ptr->data);
                ptr=ptr->next;
            }
        }
    }

    void search(){
        int x,i=1,f=0;
        if(head==NULL){
            printf("List Empty");
        }
        else{
            printf("Enter the item : ");
            scanf("%d",&x);
            for(ptr=head; ptr!=NULL; ptr=ptr->next){
                if(ptr->data==x){
                    printf("Element found at node %d",i);
```

```c
            f=1;
          }
          i++;
        }
      if(f==0){
         printf("Element not found");
      }
    }
}

void main(){
   int ch;
   do{
      printf("\n1. Insert at front\n2. Insert at Specific Position\n3. Delete at front\n4. Delete at rear\n5. Search\n6. Display\n7. Exit\nEnter your choice(1-7) : ");
      scanf("%d",&ch);
      switch(ch){
         case 1: ins_beg();
               break;
         case 2: ins_spec();
               break;
         case 3: del_beg();
               break;
         case 4: del_end();
               break;
         case 5: search();
               break;
         case 6: display();
               break;
      }
   }while(ch>0&&ch<7);
}
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM12.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM12.c

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 1
Enter the item : 10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 1
Enter the item : 20

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 1
Enter the item : 30

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 30     20      10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 2
Enter the item and it's position : 25 1

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 30     25      20      10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 2
Enter the item and it's position : 15 3

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 30     25      20      15      10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 5
Enter the item : 20
Element found at node 3

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 3
30 is deleted

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6

Linked List : 25     20     15     10

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 4
10 is deleted

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 6
Linked List : 25     20     15

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 5
Enter the item : 30
Element not found

1. Insert at front
2. Insert at Specific Position
3. Delete at front
4. Delete at rear
5. Search
6. Display
7. Exit
Enter your choice(1-7) : 7

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 13**                                              **Date: 20.10.2023**

## Doubly Linked List Operations

## Aim:

13. To implement the following operations on a singly linked list
    a. Creation
    b. Count the number of nodes
    c. Insert a new node at front
    d. Insert an element at end
    e. Deletion from beginning
    f. Deletion from the end
    g. Searching
    h. Traversal.

### Algorithm:

**main()**

1. Start
2. struct node{
           int data;
           struct node *l, *r;
       }*head, *ptr, *temp;
       c=0
3. Display choices.
4. Read option ch.
       a. if ch==1 call ins_beg().
       b. if ch==2 call ins_end().
       c. if ch==3 call del_beg()
       d. if ch==4 call del_end()
       e. if ch==5 call search()
       f. if ch==6 print c
       g. if ch==7 call display()
5. Repeat step 3 while ch>0&&ch<8.
6. Stop.

**void ins_beg()**

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. c++

      if head==NULL

           ptr->r=ptr->l=NULL;

           head=ptr

      else

           ptr->l=NULL;

           ptr->r=head;

           head=ptr;

5. Exit

**void ins_end()**

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. c++

      if head==NULL

           ptr->r=ptr->l=NULL;

           head=ptr

      else

           temp=head;

           while(temp->r!=NULL){

                temp=temp->r;

           }

           temp->r=ptr;

           ptr->l=temp;

           ptr->r=NULL;

5. Exit

**void del_beg()**

1. Start
2. if head==NULL print List Empty
3. else

      c--

      print head->data is deleted

      if(head->r==NULL){

           free(head);

           head=NULL;

```
                }
                else{
                        ptr=head;
                        head=head->r;
                        head->l=NULL;
                        free(ptr);
                }
```
4. Exit.

**void del_end()**

1. Start
2. if head==NULL print List Empty
3. else
```
                c--;
                if(head->r==NULL){
                        printf("%d is deleted",head->data);
                        free(head);
                        head=NULL;
                }
                else{
                        ptr=head;
                        while(ptr->r!=NULL){
                                ptr=ptr->r;
                        }
                        printf("%d is deleted",ptr->data);
                        ptr->l->r=NULL;
                        free(ptr);
                }
```
4. Exit.

**void display()**

1. Start
2. if head==NULL print List Empty
3. else
```
                printf("Linked List : ");
                while(ptr!=NULL){
                        printf("%d\t",ptr->r);
                        ptr=ptr->r;
                }
```
4. Exit.

**void search()**

1. Start
2. Declare x,i=1,f=0
3. if head==NULL print List Empty
4. else

       read x
       for(ptr=head; ptr!=NULL; ptr=ptr->r){
           if(ptr->r==x){
               print element found at node i
               set f=1
           }
           i++
       }
       if f ==0 print Element not found

5. Exit.

## Program

```c
#include<stdio.h>
#include<stdlib.h>
int c=0;
struct node{
   int data;
   struct node *l, *r;
}*head, *ptr, *temp;

void ins_beg(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   c++;
   if(head==NULL){
     ptr->r=ptr->l=NULL;
     head=ptr;
   }
   else{
     ptr->l=NULL;
     ptr->r=head;
     head=ptr;
   }
}
```

```c
void ins_end(){
    ptr =  malloc(sizeof(struct node));
    printf("Enter the item : ");
    scanf("%d",&ptr->data);
    c++;
    if(head==NULL){
        ptr->r=ptr->l=NULL;
        head=ptr;
    }
    else{
        temp=head;
        while(temp->r!=NULL){
            temp=temp->r;
        }
        temp->r=ptr;
        ptr->l=temp;
        ptr->r=NULL;
    }
}

void del_beg(){
    if(head==NULL){
        printf("List Empty");
    }
    else{
        c--;
        printf("%d is deleted",head->data);
        if(head->r==NULL){
            free(head);
            head=NULL;
        }
        else{
            ptr=head;
            head=head->r;
            head->l=NULL;
            free(ptr);
        }
    }
}

void del_end(){
    if(head==NULL){
```

```c
        printf("List Empty");
      }
      else{
        c--;
        if(head->r==NULL){
          printf("%d is deleted",head->data);
          free(head);
          head=NULL;
        }
        else{
          ptr=head;
          while(ptr->r!=NULL){
            ptr=ptr->r;
          }
          printf("%d is deleted",ptr->data);
          ptr->l->r=NULL;
          free(ptr);
        }
      }
    }

    void display(){
      ptr=head;
      if(ptr==NULL){
        printf("List Empty");
      }
      else{
        printf("Doubly Linked List : ");
        while(ptr!=NULL){
          printf("%d\t",ptr->data);
          ptr=ptr->r;
        }
      }
    }

    void search(){
      int x,i=1,f=0;
      if(head==NULL){
        printf("List Empty");
      }
      else{
        printf("Enter the item : ");
```

```c
        scanf("%d",&x);
        for(ptr=head; ptr!=NULL; ptr=ptr->r){
            if(ptr->data==x){
                printf("Element found at node %d",i);
                f=1;
            }
            i++;
        }
        if(f==0){
            printf("Element not found");
        }
    }
}

void main(){
    int ch;
    do{
        printf("\n1. Insert at front\n2. Insert at rear\n3. Delete at front\n4. Delete at
rear\n5. Display\n6. Search\n7. Count\n8. Exit\nEnter your choice(1-8) : ");
        scanf("%d",&ch);
        switch(ch){
            case 1: ins_beg();
                    break;
            case 2: ins_end();
                    break;
            case 3: del_beg();
                    break;
            case 4: del_end();
                    break;
            case 5: display();
                    break;
            case 6: search();
                    break;
            case 7: printf("Number of nodes : %d",c);
                    break;
        }
    }while(ch>0&&ch<8);
}
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM13.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM13.c

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 1
Enter the item : 10

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 1
Enter the item : 20

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 1
Enter the item : 30

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 5
Doubly Linked List : 30   20      10


1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 2
Enter the item : 40


1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 2
Enter the item : 50

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 5
Doubly Linked List : 30   20      10      40      50

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 6
Enter the item : 10

Element found at node 3
1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 7
Number of nodes : 5

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 3
30 is deleted


1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 5
Doubly Linked List : 20  10     40     50


1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 4
50 is deleted

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 5
Doubly Linked List : 20   10      40

1. Insert at front
2. Insert at rear
3. Delete at front
4. Delete at rear
5. Display
6. Search
7. Count
8. Exit
Enter your choice(1-8) : 8

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 14**                                    **Date: 27.10.2023**

## Linked Stack Operations

**Aim:**

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

**Algorithm:**

**main()**

1. Start
2. struct node{
           int data;
           struct node *next;
   }*top, *ptr;
3. Display choices.
4. Read option ch.
       a. if ch==1 call push().
       b. if ch==2 call pop().
       c. if ch==3 call display()
5. Repeat step 3 while ch>0&&ch<4.
6. Stop.

**void push()**

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. ptr->next=top; top=ptr;
5. Exit

**void pop**()

1. Start
2. if head==NULL print Stack Underflow
3. else

     ptr=top
     print ptr->data is deleted
     top=top->next;
     free(ptr);

4. Exit.

**void display**()

1. Start
2. if head==NULL print Stack Empty
3. else

     while(ptr!=NULL){
        print ptr->data
        ptr=ptr->next
     }

4. Exit.

## **Program**

```
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct node *next;
}*top, *ptr;

void push(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   ptr->next=top;
   top=ptr;
}

void pop(){
   if(top==NULL){
      printf("Stack Underflow");
   }
   else{
```

```c
        ptr=top;
        printf("%d is deleted",ptr->data);
        top=top->next;
        free(ptr);
    }
}

void display(){
    ptr=top;
    if(ptr==NULL){
        printf("Stack Empty");
    }
    else{
        printf("Stack : ");
        while(ptr!=NULL){
            printf("%d\t",ptr->data);
            ptr=ptr->next;
        }
    }
}

void main(){
    int ch;
    do{
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\nEnter your choice(1-4) : ");
        scanf("%d",&ch);
        switch(ch){
            case 1: push();
                    break;
            case 2: pop();
                    break;
            case 3: display();
                    break;
        }
    }while(ch>0&&ch<4);
}
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM14.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM14.c

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 20

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 30

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 3
Stack : 30    20     10

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
30 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
20 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 2
10 is deleted

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 3
Stack Empty

1. Push
2. Pop
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$

## Experiment 15                                          Date: 27.10.2023

## Linked Queue Operations

### Aim:

15. To implement a menu driven program to perform following queue operations using linked list
    a. Enqueue
    b. Dequeue
    c. Traversal

### Algorithm:

**main()**

1. Start
2. struct node{
           int data;
           struct node *next;
       }*top, *ptr, *f, *r;
3. Display choices.
4. Read option ch.
       a. if ch==1 call enqueue().
       b. if ch==2 call dequeue().
       c. if ch==3 call display()
5. Repeat step 3 while ch>0&&ch<4.
6. Stop.

**void enqueue()**

1. Start
2. ptr = malloc(sizeof(struct node))
3. Read ptr->data
4. if f==NULL
           f=r=ptr;
       else
           r->next=ptr;
           r=ptr;
5. Exit

**void dequeue()**

1. Start
2. if f==NULL print Queue is empty
3. else

    ptr=f

    print ptr->data is deleted

    f=ptr->next;

    free(ptr);
4. Exit.

**void display()**

1. Start
2. if head==NULL print Queue is empty
3. else

    while(ptr!=NULL){

        print ptr->data

        ptr=ptr->next

    }
4. Exit.

## Program

```c
#include<stdio.h>
#include<stdlib.h>
struct node{
   int data;
   struct node *next;
}*top, *ptr, *f, *r;

void enqueue(){
   ptr = malloc(sizeof(struct node));
   printf("Enter the item : ");
   scanf("%d",&ptr->data);
   if(f==NULL){
      f=r=ptr;
   }
   else{
      r->next=ptr;
      r=ptr;
   }
}
```

```c
    void dequeue(){
      if(f==NULL){
        printf("Queue is empty");
      }
      else{
        ptr=f;
        printf("%d is deleted",ptr->data);
        f=ptr->next;
        free(ptr);
      }
    }


    void display(){
      if(f==NULL){
        printf("Queue is empty");
      }
      else{
        ptr=f;
        printf("Queue : ");
        while(ptr!=NULL){
          printf("%d\t",ptr->data);
          ptr=ptr->next;
        }
      }
    }

    void main(){
      int ch;
      do{
        printf("\n1.  Enqueue\n2.  Dequeue\n3.  Display\n4.  Exit\nEnter  your
    choice(1-4) : ");
        scanf("%d",&ch);
        switch(ch){
          case 1: enqueue();
                break;
          case 2: dequeue();
                break;
          case 3: display();
                break;
        }
      }while(ch>0&&ch<4);
    }
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM15.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM15.c

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 10

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 20

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 1
Enter the item : 30

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 3
Queue : 10   20     30

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 2
10 is deleted

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 2
20 is deleted

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 2
30 is deleted

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 3
Queue is empty

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice(1-4) : 4

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 16**                                              **Date: 02.11.2023**

## Binary Search Tree Operations

**Aim:**

16. Menu Driven program to implement Binary Search Tree (BST) and to perform following operations
    a. Insertion of a node.
    b. Deletion of a node.
    c. In-order traversal.
    d. Pre-order traversal.
    e. Post-order traversal.

**Algorithm:**

**main()**

1. Start
2. struct node{
       int data;
       struct node *l,*r;
   }*root, *ptr, *succ, *succparent;
3. Declare ch and x
4. Display choices.
5. Read option ch.
       a. if ch==1 read x and call root=insert(root,x).
       b. if ch==2 read x and call root=del(root,x).
       c. if ch==3 call inorder(root)
       d. if ch==4 call preorder(root)
       e. if ch==5 call postorder(root)
6. Repeat step 3 while ch>0&&ch<6.
7. Stop.

**struct node* create(int x)**

1. Start
2. ptr=malloc(sizeof(struct node));
3. ptr->data=x;
4. ptr->l=ptr->r=NULL;
5. return ptr;
6. Exit

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

1. Start
2. if root==NULL return create(x)
3. if x>root->data
        root->r=insert(root->r,x);
   else
        root->l=insert(root->l,x);
4. return root;
5. Exit.

**struct node\* del(struct node\* root, int x)**

1. Start
2. if root==NULL return root
3. if x>root->data
        root->r=del(root->r,x)
        return root
   else if x<root->data
        root->l=del(root->l,x)
        return root
4. if root->l==NULL
        ptr=root->r
        free(root)
        return ptr
   else if root->r==NULL
        ptr=root->l
        free(root)
        return ptr
5. succparent=root
   succ=root->r;
   while(succ->l!=NULL){
        succparent=succ;
        succ=root->l;
   }
6. if succparent!=root
        succparent->l=succ->r
   else
        succparent->r=succ->r
7. root->data=succ->data
8. free(succ);
9. return root;
10. Exit.

**void inorder(struct node\* root)**

1. Start
2. if(root!=NULL)

     inorder(root->l)

     print root->data

     inorder(root->r)
3. Exit.

**void preorder(struct node\* root)**

1. Start
2. if(root!=NULL)

     print root->data

     inorder(root->l)

     inorder(root->r)
3. Exit.

**void postorder(struct node\* root)**

1. Start
2. if(root!=NULL)

     inorder(root->l)

     inorder(root->r)

     print root->data
3. Exit.

## Program

```
#include<stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node *l,*r;
}*root, *ptr, *succ, *succparent;

struct node* create(int x){
   ptr=malloc(sizeof(struct node));
   ptr->data=x;
   ptr->l=ptr->r=NULL;
   return ptr;
}
```

```
struct node* insert(struct node* root, int x){
   if(root==NULL){
      return create(x);
   }
   if(x>root->data){
      root->r=insert(root->r,x);
   }
   else{
      root->l=insert(root->l,x);
   }

   return root;
}

struct node* del(struct node* root, int x){
   if(root==NULL){
      return root;
   }

   if(x>root->data){
      root->r=del(root->r,x);
      return root;
   }
   else if(x<root->data){
      root->l=del(root->l,x);
      return root;
   }

   if(root->l==NULL){
      ptr=root->r;
      free(root);
      return ptr;
   }
   else if(root->r==NULL){
      ptr=root->l;
      free(root);
      return ptr;
   }
```

```c
      succparent=root;
      succ=root->r;
      while(succ->l!=NULL){
         succparent=succ;
         succ=root->l;
      }

      if(succparent!=root){
         succparent->l=succ->r;
      }
      else{
         succparent->r=succ->r;
      }

      root->data=succ->data;

      free(succ);
      return root;
   }

   void inorder(struct node* root){
      if(root!=NULL){
         inorder(root->l);
         printf("%d\t",root->data);
         inorder(root->r);
      }
   }

   void preorder(struct node* root){
      if(root!=NULL){
         printf("%d\t",root->data);
         inorder(root->l);
         inorder(root->r);
      }
   }
```

```
void postorder(struct node* root){
    if(root!=NULL){
        inorder(root->l);
        inorder(root->r);
        printf("%d\t",root->data);
    }
}
void main(){
    int ch,x;
    do{
        printf("\n1.  Insert\n2.  Delete\n3.  Inorder  Traversal\n4.  Preorder Traversal\n5. Postorder Traversal\n6. Exit\nEnter your choice(1-6) : ");
        scanf("%d",&ch);
        switch(ch){
            case 1: printf("Enter the element : ");
                scanf("%d",&x);
                root=insert(root,x);
                break;
            case 2: printf("Enter the element : ");
                scanf("%d",&x);
                root=del(root,x);
                break;
            case 3: printf("Inorder Traversal : ");
                inorder(root);
                break;
            case 4: printf("Preorder Traversal : ");
                preorder(root);
                break;
            case 5: printf("Postorder Traversal : ");
                postorder(root);
                break;
        }
    }while(ch>0&&ch<6);
}
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM16.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM16.c

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 1
Enter the element : 10

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 1
Enter the element : 5

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 1
Enter the element : 15

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 3
Inorder Traversal : 5       10      15

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 4
Preorder Traversal : 10    5        15

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 5
Postorder Traversal : 5    15      10

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 2
Enter the element : 15

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 3
Inorder Traversal : 5        10

1. Insert
2. Delete
3. Inorder Traversal
4. Preorder Traversal
5. Postorder Traversal
6. Exit
Enter your choice(1-6) : 6

mits@mits:~/Desktop/S1MCA/ADS_lab$

**Experiment 17**                                        **Date: 09.11.2023**

## Bitstring Operations

## Aim:

17. To implement set operations using bit strings.

### Algorithm:

**main()**

1. Start
2. Declare int a[11], b[11], res[11], U[11]={1,2,3,4,5,6,7,8,9,10},s1,s2,ch;
3. Read size of bit-string 1 s1
4. Call input(a,s1) and display(a)
5. Read size of bit-string 2 s2
6. Call input(b,s2) and display(b)
7. Display choices.
8. Read option ch.
   a. if ch==1 call set_union().
   b. if ch==2 call set_intersection().
   c. if ch==3 call set_difference().
   d. if ch==3 if(set_equality())

                        print Bit strings are equal.

                else

                        print Bit strings are not equal.
9. Repeat step 3 while ch>0&&ch<4.
10. Stop.

**void set_union()**

1. Start
2. for(int i=1;i<11;i++)
            res[i]=a[i] | b[i];
3. display(res)
4. Exit.

**void set_intersection()**

1. Start
2. for(int i=1;i<11;i++)
            res[i]=a[i] & b[i];
3. display(res)
4. Exit.

**void set_union()**

1. Start
2. for(int i=1;i<11;i++)
            res[i]=a[i] & ~b[i];
3. display(res)
4. Exit.

**bool set_equality()**

1. Start
2. for(int i=1;i<11;i++)
            if a[i] != b[i]
                    return false
3. return true
4. Exit.

**void input(int bs[], int n)**

1. Start
2. Declare x
3. for(int i=1;i<11;i++)
            read x
            bs[x]=1
4. Exit.

**void display(int bs[])**

1. Start
2. for(int i=1;i<11;i++)
            print bs[i]
3. Exit.

## Program

```c
#include<stdio.h>
#include <stdbool.h>
int a[11], b[11], res[11];
int U[11]={1,2,3,4,5,6,7,8,9,10};

void display(int bs[]){
   for(int i=1;i<11;i++){
     printf("%d\t",bs[i]);
   }
}
```

```c
void input(int bs[], int n){
    int x;
    printf("Enter the elements : ");
    for(int i=0;i<n;i++){
        scanf("%d",&x);
        bs[x]=1;
    }
}

void set_union(){
    for(int i=1;i<11;i++){
        res[i]=a[i] | b[i];
    }

    printf("\nUnion Set : ");
    display(res);
}

void set_intersection(){
    for(int i=1;i<11;i++){
        res[i]=a[i] & b[i];
    }

    printf("\nIntersection Set : ");
    display(res);
}

void set_difference(){
    for(int i=1;i<11;i++){
        res[i]=a[i] & ~b[i];
    }

    printf("\nDifference Set : ");
    display(res);
}
```

```
bool set_equality(){
    for(int i=1;i<11;i++){
        if(a[i] != b[i]){
            return false;
        }
    }
    return true;
}
```

**Output**

mits@mits:~/Desktop/S1MCA/ADS_lab$ gcc PGM17.c
mits@mits:~/Desktop/S1MCA/ADS_lab$ ./a.out PGM17.c

Enter the size of bit-string 1 : 5
Enter the elements : 1 3 5 7 9
Set A : 1      0      1      0      1      0      1      0      1      0

Enter the size of bit-string 2 : 5
Enter the elements : 2 4 6 8 10
Set B : 0      1      0      1      0      1      0      1      0      1

1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 1
Union Set : 1      1      1      1      1      1      1      1      1      1

1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 2
Intersection Set : 0 0      0      0      0      0      0      0      0      0

1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 3
Difference Set : 1  0     1     0     1     0     1     0     1     0


1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 4
Bit strings are not equal


1. Union
2. Intersection
3. Difference
4. Equality
5. Exit
Enter your choice : 5

mits@mits:~/Desktop/S1MCA/ADS_lab$