Experiment -1

Simulate the following CPU scheduling algorithms:

(a) Round Robin (b) SJF (c) FCFS (d) Priority

(a) Round Robin

```
#include<conio.h>
 void main()
{
   // initlialize the variable name
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");  // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time qunat
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
```

```
    {
        temp[i] = temp[i] - quant;
        sum = sum + quant;
    }
    if(temp[i]==0 && count==1)
    {
        y--; //decrement the process no.
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);
        wt = wt+sum-at[i]-bt[i];
        tat = tat+sum-at[i];
        count =0;
    }
    if(i==NOP-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
// represents the average waiting time and Turn Around time
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
//getch();
}
```

Output:-

```
Total number of process in the system: 6
 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       0
Burst time is:  7
 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       1
```

Burst time is: 4

Enter the Arrival and Burst time of the Process[3]

Arrival time is: 2

Burst time is: 15

Enter the Arrival and Burst time of the Process[4]

Arrival time is: 3

Burst time is: 11

Enter the Arrival and Burst time of the Process[5]

Arrival time is: 4

Burst time is: 20

Enter the Arrival and Burst time of the Process[6]

Arrival time is: 4

Burst time is: 9

Enter the Time Quantum for the process: 5

Process No	Burst Time	TAT	Waiting Time
Process No[2]	4	8	4
Process No[1]	7	31	24
Process No[6]	9	46	37
Process No[3]	15	53	38
Process No[4]	11	53	42
Process No[5]	20	62	42

Average Turn Around Time: 31.166666

Average Waiting Time: 42.166668

b) SJF

```
#include<stdio.h>
void main()
{
int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
float avg_wt,avg_tat;
printf("Enter number of process:");
scanf("%d",&n);
printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
```

```
p[i]=i+1;
}
for(i=0;i<n;i++)
{
pos=i;
for(j=i+1;j<n;j++)
{
if(bt[j]<bt[pos])
pos=j;
}
temp=bt[i];
bt[i]=bt[pos];
bt[pos]=temp;
temp=p[i];
p[i]=p[pos];
p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=0;
for(j=0;j<i;j++)
wt[i]+=bt[j];
total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
tat[i]=bt[i]+wt[i];
total+=tat[i];
printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output:-

Enter number of process:4
Enter Burst Time:
p1:4
p2:1
p3:8
p4:1

Process	Burst Time	Waiting Time	Turnaround Time
p2	1	0	1
p4	1	1	2
p1	4	2	6
p3	8	6	14

Average Waiting Time=2.250000
Average Turnaround Time=5.750000

c) FCFS

```
#include <stdio.h>

struct Process {
   int pid;          // Process ID
   int burstTime;    // Burst Time
   int arrivalTime;  // Arrival Time
   int waitTime;     // Waiting Time
   int turnAroundTime;  // Turnaround Time
};

// Function to find waiting time for each process
void findWaitingTime(struct Process proc[], int n) {
         int i;
   int serviceTime[n];  // Store cumulative burst time for service
   serviceTime[0] = proc[0].arrivalTime;  // First process starts when it arrives
   proc[0].waitTime = 0;  // First process has no waiting time

   for (i = 1; i < n; i++) {
      // Cumulative burst time
      serviceTime[i] = serviceTime[i - 1] + proc[i - 1].burstTime;

      // Waiting time = service time - arrival time
```

```
        proc[i].waitTime = serviceTime[i] - proc[i].arrivalTime;

        // If waiting time is negative, set it to 0 (no waiting)
        if (proc[i].waitTime < 0)
            proc[i].waitTime = 0;
    }
}

// Function to find turnaround time for each process
void findTurnAroundTime(struct Process proc[], int n) {
            int i;
    for (i = 0; i < n; i++) {
        proc[i].turnAroundTime = proc[i].burstTime + proc[i].waitTime;
    }
}

// Function to sort processes by arrival time
void sortProcessesByArrival(struct Process proc[], int n) {
            int i,j;
    struct Process temp;
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (proc[i].arrivalTime > proc[j].arrivalTime) {
                temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }
}

// Function to calculate average waiting and turnaround time
void findAverageTime(struct Process proc[], int n) {
            int i;
    findWaitingTime(proc, n);
    findTurnAroundTime(proc, n);

    int totalWaitTime = 0, totalTurnAroundTime = 0;

    printf("Process\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
```

```
    for (i = 0; i < n; i++) {
        totalWaitTime += proc[i].waitTime;
        totalTurnAroundTime += proc[i].turnAroundTime;

        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].pid, proc[i].arrivalTime, proc[i].burstTime,
proc[i].waitTime, proc[i].turnAroundTime);
    }

    printf("\nAverage Waiting Time = %.2f", (float)totalWaitTime / n);
    printf("\nAverage Turnaround Time = %.2f\n", (float)totalTurnAroundTime / n);
}

int main() {
    int n,i;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process proc[n];

    for (i = 0; i < n; i++) {
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d%d", &proc[i].arrivalTime, &proc[i].burstTime);
        proc[i].pid = i + 1;  // Assign process ID
    }

    // Sort processes by arrival time
    sortProcessesByArrival(proc, n);

    // Find average time and display results
    findAverageTime(proc, n);

    return 0;
}
```

Output:

Enter the number of processes: 5
Enter arrival time and burst time for process 1: 2 2
Enter arrival time and burst time for process 2: 5 6

```
Enter arrival time and burst time for process 3: 0 4
Enter arrival time and burst time for process 4: 0 7
Enter arrival time and burst time for process 5: 7 4
Process Arrival Time    Burst Time      Waiting Time    Turnaround Time
3           0           4           0           4
4           0           7           4           11
1           2           2           9           11
2           5           6           8           14
5           7           4           12           16

Average Waiting Time = 6.60
Average Turnaround Time = 11.20
```

d) Priority

```
#include <stdio.h>

struct Process {
    int pid;          // Process ID
    int burstTime;    // Burst Time
    int arrivalTime;  // Arrival Time
    int priority;     // Priority
    int waitTime;     // Waiting Time
    int turnAroundTime;  // Turnaround Time
    int completionTime;  // Completion Time
};

// Function to sort processes by arrival time and priority
void sortProcesses(struct Process proc[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (proc[i].arrivalTime > proc[j].arrivalTime ||
                (proc[i].arrivalTime == proc[j].arrivalTime && proc[i].priority > proc[j].priority)) {
                temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }
}
```

```
// Function to find waiting time for each process
void findWaitingTime(struct Process proc[], int n) {
    int serviceTime[n];  // To store cumulative service times
    serviceTime[0] = proc[0].arrivalTime;  // First process starts at its arrival time
    proc[0].waitTime = 0;  // First process has no waiting time

    for (int i = 1; i < n; i++) {
        serviceTime[i] = serviceTime[i - 1] + proc[i - 1].burstTime;  // Cumulative burst time
        proc[i].waitTime = serviceTime[i] - proc[i].arrivalTime;  // Waiting time = service time -
arrival time

        // If waiting time is negative, set it to 0 (CPU waits for process to arrive)
        if (proc[i].waitTime < 0)
            proc[i].waitTime = 0;
    }
}

// Function to find turnaround time for each process
void findTurnAroundTime(struct Process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].turnAroundTime = proc[i].burstTime + proc[i].waitTime;
        proc[i].completionTime = proc[i].turnAroundTime + proc[i].arrivalTime;
    }
}

// Function to calculate average waiting and turnaround time
void findAverageTime(struct Process proc[], int n) {
    findWaitingTime(proc, n);
    findTurnAroundTime(proc, n);

    int totalWaitTime = 0, totalTurnAroundTime = 0;

    printf("Process\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround
Time\tCompletion Time\n");

    for (int i = 0; i < n; i++) {
        totalWaitTime += proc[i].waitTime;
        totalTurnAroundTime += proc[i].turnAroundTime;
```

```
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",
               proc[i].pid, proc[i].arrivalTime, proc[i].burstTime, proc[i].priority,
               proc[i].waitTime, proc[i].turnAroundTime, proc[i].completionTime);
    }

    printf("\nAverage Waiting Time = %.2f", (float)totalWaitTime / n);
    printf("\nAverage Turnaround Time = %.2f\n", (float)totalTurnAroundTime / n);
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process proc[n];

    // Input details for each process
    for (int i = 0; i < n; i++) {
        printf("Enter arrival time, burst time, and priority for process %d: ", i + 1);
        scanf("%d%d%d", &proc[i].arrivalTime, &proc[i].burstTime, &proc[i].priority);
        proc[i].pid = i + 1;  // Assign process ID
    }

    // Sort processes based on arrival time and priority
    sortProcesses(proc, n);

    // Calculate average time and display results
    findAverageTime(proc, n);

    return 0;
}
```

Output:

```
Enter the number of processes: 5
Enter arrival time, burst time, and priority for process 1: 0
4
2
Enter arrival time, burst time, and priority for process 2: 1
3
```

3
Enter arrival time, burst time, and priority for process 3: 2
1
4
Enter arrival time, burst time, and priority for process 4: 3
5
5
Enter arrival time, burst time, and priority for process 5: 4
2
5

Process	Arrival Time	Burst Time	Priority	Waiting Time	Turnaround Time	Completion Time
1	0	4	2	0	4	4
2	1	3	3	3	6	7
3	2	1	4	5	6	8
4	3	5	5	5	10	13
5	4	2	5	9	11	15

Average Waiting Time = 4.40
Average Turnaround Time = 7.40

Experiment - 2

Simulate the following page replacement algorithms:
a) FIFO
b) LRU
c) LFU

a) FIFO:

```
#include <stdio.h>
int main() {
    int pages[30], frames[10], pageFaults = 0, m, n, s, pagesSize, frameSize;
    int counter = 0, flag1, flag2;

    printf("Enter the number of pages: ");
    scanf("%d", &pagesSize);

    printf("Enter the reference string (page numbers):\n");
    for (m = 0; m < pagesSize; m++) {
        scanf("%d", &pages[m]);
```

```
}

printf("Enter the number of frames: ");
scanf("%d", &frameSize);

for (m = 0; m < frameSize; m++) {
    frames[m] = -1;  // Initialize all frames to -1 (indicating empty)
}

printf("\nPage replacement process:\n");

for (n = 0; n < pagesSize; n++) {
    flag1 = flag2 = 0;

    // Check if the page is already in a frame
    for (m = 0; m < frameSize; m++) {
        if (frames[m] == pages[n]) {
            flag1 = flag2 = 1;
            break;
        }
    }

    // If the page is not in any frame
    if (flag1 == 0) {
        // Replace the oldest page (FIFO) in the frame
        frames[counter] = pages[n];
        counter = (counter + 1) % frameSize;
        pageFaults++;

        // Print the current state of frames
        printf("Page %d: ", pages[n]);
        for (m = 0; m < frameSize; m++) {
            if (frames[m] != -1) {
                printf("%d ", frames[m]);
            } else {
                printf("- ");
            }
        }
        printf("(Page Fault)\n");
    } else {
```

```
            printf("Page %d: No Page Fault\n", pages[n]);
        }
    }

    printf("\nTotal Page Faults = %d\n", pageFaults);

    return 0;
}
```

OUTPUT:-

```
Enter the number of pages: 20
Enter the reference string (page numbers):
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the number of frames: 3
Page replacement process:
Page 7: 7 - - (Page Fault)
Page 0: 7 0 - (Page Fault)
Page 1: 7 0 1 (Page Fault)
Page 2: 2 0 1 (Page Fault)
Page 0: No Page Fault
Page 3: 2 3 1 (Page Fault)
Page 0: 2 3 0 (Page Fault)
Page 4: 4 3 0 (Page Fault)
Page 2: 4 2 0 (Page Fault)
Page 3: 4 2 3 (Page Fault)
Page 0: 0 2 3 (Page Fault)
Page 3: No Page Fault
Page 2: No Page Fault
Page 1: 0 1 3 (Page Fault)
Page 2: 0 1 2 (Page Fault)
Page 0: No Page Fault
Page 1: No Page Fault
Page 7: 7 1 2 (Page Fault)
Page 0: 7 0 2 (Page Fault)
Page 1: 7 0 1 (Page Fault)

Total Page Faults = 15
```

b) <u>OPTIMAL (LFU)</u>

```
#include <stdio.h>

int findOptimal(int pages[], int frames[], int frameSize, int currentIndex, int pagesSize) {
    int farthest = currentIndex, pos = -1;

    for (int i = 0; i < frameSize; i++) {
        int j;
        for (j = currentIndex; j < pagesSize; j++) {
            if (frames[i] == pages[j]) {
                if (j > farthest) {
                    farthest = j;
                    pos = i;
                }
                break;
            }
        }

        // If the frame is never used again, return its position
        if (j == pagesSize) {
            return i;
        }
    }

    return (pos == -1) ? 0 : pos;
}

int main() {
    int pages[30], frames[10], pageFaults = 0, pagesSize, frameSize, flag1, flag2;

    printf("Enter the number of pages: ");
    scanf("%d", &pagesSize);

    printf("Enter the reference string (page numbers):\n");
    for (int i = 0; i < pagesSize; i++) {
        scanf("%d", &pages[i]);
    }

    printf("Enter the number of frames: ");
    scanf("%d", &frameSize);
```

```c
for (int i = 0; i < frameSize; i++) {
    frames[i] = -1;  // Initialize all frames to -1 (indicating empty)
}

printf("\nPage replacement process:\n");

for (int i = 0; i < pagesSize; i++) {
    flag1 = flag2 = 0;

    // Check if the page is already in a frame
    for (int j = 0; j < frameSize; j++) {
        if (frames[j] == pages[i]) {
            flag1 = flag2 = 1;
            break;
        }
    }

    // If the page is not in any frame
    if (flag1 == 0) {
        // If there's an empty frame, use it
        for (int j = 0; j < frameSize; j++) {
            if (frames[j] == -1) {
                frames[j] = pages[i];
                flag2 = 1;
                pageFaults++;
                break;
            }
        }
    }

    // If no empty frame, replace using the optimal strategy
    if (flag2 == 0) {
        int pos = findOptimal(pages, frames, frameSize, i + 1, pagesSize);
        frames[pos] = pages[i];
        pageFaults++;
    }

    // Print the current state of frames
    printf("Page %d: ", pages[i]);
```

```
        for (int j = 0; j < frameSize; j++) {
            if (frames[j] != -1) {
                printf("%d ", frames[j]);
            } else {
                printf("- ");
            }
        }

        if (flag1 == 0) {
            printf("(Page Fault)\n");
        } else {
            printf("\n");
        }
    }

    printf("\nTotal Page Faults = %d\n", pageFaults);

    return 0;
}
```

OUTPUT:-

Enter the number of pages: 20
Enter the reference string (page numbers):
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the number of frames: 3
Page replacement process:
Page 7: 7 - - (Page Fault)
Page 0: 7 0 - (Page Fault)
Page 1: 7 0 1 (Page Fault)
Page 2: 2 0 1 (Page Fault)
Page 0: 2 0 1
Page 3: 2 0 3 (Page Fault)
Page 0: 2 0 3
Page 4: 2 4 3 (Page Fault)
Page 2: 2 4 3
Page 3: 2 4 3
Page 0: 2 0 3 (Page Fault)
Page 3: 2 0 3
Page 2: 2 0 3
Page 1: 2 0 1 (Page Fault)

Page 2: 2 0 1
Page 0: 2 0 1
Page 1: 2 0 1
Page 7: 7 0 1 (Page Fault)
Page 0: 7 0 1
Page 1: 7 0 1

Total Page Faults = 9

c) <u>LRU:-</u>

```
#include <stdio.h>

int findLRU(int time[], int n) {
    int i, minimum = time[0], pos = 0;

    for (i = 1; i < n; ++i) {
        if (time[i] < minimum) {
            minimum = time[i];
            pos = i;
        }
    }

    return pos;
}

int main() {
    int noOfFrames, noOfPages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, pageFaults = 0;

    printf("Enter the number of frames: ");
    scanf("%d", &noOfFrames);

    printf("Enter the number of pages: ");
    scanf("%d", &noOfPages);

    printf("Enter the reference string (page numbers):\n");
    for (i = 0; i < noOfPages; ++i) {
        scanf("%d", &pages[i]);
    }
```

```c
for (i = 0; i < noOfFrames; ++i) {
    frames[i] = -1;  // Initialize all frames to -1 (indicating empty)
}

printf("\nPage replacement process:\n");

for (i = 0; i < noOfPages; ++i) {
    flag1 = flag2 = 0;

    // Check if the page is already in a frame
    for (j = 0; j < noOfFrames; ++j) {
        if (frames[j] == pages[i]) {
            counter++;
            time[j] = counter;  // Update the time of access
            flag1 = flag2 = 1;
            break;
        }
    }

    // If the page is not in a frame
    if (flag1 == 0) {
        for (j = 0; j < noOfFrames; ++j) {
            if (frames[j] == -1) {  // If there's an empty frame, use it
                counter++;
                pageFaults++;
                frames[j] = pages[i];
                time[j] = counter;
                flag2 = 1;
                break;
            }
        }
    }

    // If no empty frame, replace the least recently used page
    if (flag2 == 0) {
        pos = findLRU(time, noOfFrames);
        counter++;
        pageFaults++;
        frames[pos] = pages[i];
        time[pos] = counter;
```

```
        }

        // Print the current state of frames
        printf("Page %d: ", pages[i]);
        for (j = 0; j < noOfFrames; ++j) {
            if (frames[j] != -1) {
                printf("%d ", frames[j]);
            } else {
                printf("- ");
            }
        }

        if (flag1 == 0) {
            printf("(Page Fault)\n");
        } else {
            printf("\n");
        }
    }

    printf("\nTotal Page Faults = %d\n", pageFaults);

    return 0;
}
```

OUTPUT:-

Enter the number of frames: 3
Enter the number of pages: 20
Enter the reference string (page numbers):
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Page replacement process:
Page 7: 7 - - (Page Fault)
Page 0: 7 0 - (Page Fault)
Page 1: 7 0 1 (Page Fault)
Page 2: 2 0 1 (Page Fault)
Page 0: 2 0 1
Page 3: 2 0 3 (Page Fault)
Page 0: 2 0 3
Page 4: 4 0 3 (Page Fault)

Page 2: 4 0 2 (Page Fault)
Page 3: 4 3 2 (Page Fault)
Page 0: 0 3 2 (Page Fault)
Page 3: 0 3 2
Page 2: 0 3 2
Page 1: 1 3 2 (Page Fault)
Page 2: 1 3 2
Page 0: 1 0 2 (Page Fault)
Page 1: 1 0 2
Page 7: 1 0 7 (Page Fault)
Page 0: 1 0 7
Page 1: 1 0 7

Total Page Faults = 12

Experiment -3

Write a C program that illustrates two processes communicating using shared memory

Below is a C program that demonstrates inter-process communication (IPC) using shared memory. The program consists of two processes: a writer process that writes a message to shared memory and a reader process that reads that message.

Common Functions

unistd.h

1. **Process Control:**
    - **fork(): Create a new process by duplicating the calling process.**
    - exec(): Replace the current process image with a new process image (used to run a new program).
    - getpid(): Get the process ID of the calling process.
    - **getppid(): Get the parent process ID.**
2. **File Operations:**
    - read(): Read data from a file descriptor.
    - write(): Write data to a file descriptor.
    - close(): Close a file descriptor.
3. **Working with Directories:**
    - chdir(): Change the current working directory.
    - getcwd(): Get the current working directory.
4. **Miscellaneous:**
    - sleep(): Suspend execution for a specified number of seconds.

    - usleep(): Suspend execution for a specified number of microseconds.

<sys/types.h>:

- This header defines data types used in system calls, such as pid_t, key_t, and others.

<sys/ipc.h>:

- This header includes definitions for IPC (Inter-Process Communication) mechanisms. It defines the structures and constants used for shared memory, message queues, and semaphores.

<sys/shm.h>:

- This header provides the declarations for shared memory functions, including shmget(), shmat(), shmdt(), and shmctl().

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>

#define SHM_SIZE 1024  // Size of shared memory segment

int main() {
   int shm_id;
   char *shm_ptr;

   // Create a shared memory segment
   shm_id = shmget(IPC_PRIVATE, SHM_SIZE, IPC_CREAT | 0666);
   if (shm_id < 0) {
      perror("shmget failed");
      exit(1);
   }

   // Fork a new process
   pid_t pid = fork();
```

```
if (pid < 0) {
    perror("fork failed");
    exit(1);
}

// Writer Process
if (pid == 0) {
    // Attach to the shared memory segment
    shm_ptr = (char *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (char *)(-1)) {
        perror("shmat failed");
        exit(1);
    }

    // Write data to shared memory
    const char *message = "Hello from the writer process!";
    strncpy(shm_ptr, message, SHM_SIZE);
    printf("Writer: Wrote to shared memory: %s\n", shm_ptr);

    // Detach from shared memory
    shmdt(shm_ptr);
    exit(0);
}
// Reader Process
else {
    // Wait for the writer to finish
    wait(NULL);

    // Attach to the shared memory segment
    shm_ptr = (char *)shmat(shm_id, NULL, 0);
    if (shm_ptr == (char *)(-1)) {
        perror("shmat failed");
        exit(1);
    }

    // Read data from shared memory
    printf("Reader: Read from shared memory: %s\n", shm_ptr);

    // Detach from shared memory
    shmdt(shm_ptr);
```

```
        // Destroy the shared memory segment
        shmctl(shm_id, IPC_RMID, NULL);
    }
    return 0;
}
```

Output:-

Writer: Wrote to shared memory: Hello from the writer process!

Reader: Read from shared memory: Hello from the writer process!

Experiment -4

Write a C program to simulate producer and consumer problem using semaphores

```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
void producer()
{
--mutex;
++full;
--empty;
// Item produced
x++;
printf(" \nProducer produces" "item %d", x);
++mutex;
}
void consumer()
{
--mutex;
--full;
++empty;
printf(" \nConsumer consumes ""item %d,",x);
x--;
++mutex;
}
// Driver Code
```

```
int main()
{
int n, i;
printf("\nPress 1 for Producer""\n Press 2 for Consumer""\n Press 3 for Exit");
for (i = 1; i > 0; i++) {
printf("\nEnter your choice:");
scanf("%d", &n);
// Switch Cases
switch (n) {
case 1:
if ((mutex == 1) && (empty != 0))
{
producer();
}
else {
printf("Buffer is full!");
}
break;
case 2:
if ((mutex == 1) && (full != 0))
{
consumer();
}
else {
printf("Buffer is empty!");
}
break;
case 3:
exit(0);
break;
}
}
}
```

Output:-

```
Press 1 for Producer
 Press 2 for Consumer
 Press 3 for Exit
Enter your choice:1

Producer producesitem 1
```

Enter your choice:1

Producer producesitem 2
Enter your choice:1

Producer producesitem 3
Enter your choice:1

Producer producesitem 4
Enter your choice:1

Producer producesitem 5
Enter your choice:1

Producer producesitem 6
Enter your choice:1

Producer producesitem 7
Enter your choice:1

Producer producesitem 8
Enter your choice:1

Producer producesitem 9
Enter your choice:1

Producer producesitem 10
Enter your choice:1
Buffer is full!
Enter your choice:2

Consumer consumes item 10,
Enter your choice:2

Consumer consumes item 9,
Enter your choice:2

Consumer consumes item 8,
Enter your choice:2

Consumer consumes item 7,
Enter your choice:2

Consumer consumes item 6,
Enter your choice:2

Consumer consumes item 5,
Enter your choice:2

Consumer consumes item 4,
Enter your choice:2

Consumer consumes item 3,
Enter your choice:2

Consumer consumes item 2,
Enter your choice:2

Consumer consumes item 1,
Enter your choice:2
Buffer is empty!
Enter your choice:3

<u>Experiment -5</u>

Simulate Bankers Algorithm for Dead Lock Avoidance

```
#include <stdio.h>
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k;
   n = 5;                         // Number of processes
   m = 3;                          // Number of resources
   int alloc[5][3] = {{0, 1, 0},  // P0 // Allocation Matrix
                      {2, 0, 0},  // P1
                      {3, 0, 2},  // P2
                      {2, 1, 1},  // P3
                      {0, 0, 2}}; // P4
```

```
int max[5][3] = {{7, 5, 3},  // P0 // MAX Matrix
                 {3, 2, 2},  // P1
                 {9, 0, 2},  // P2
                 {2, 2, 2},  // P3
                 {4, 3, 3}}; // P4

int avail[3] = {3, 3, 2}; // Available Resources

int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++)
{
    f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
        need[i][j] = max[i][j] - alloc[i][j];
}
int y = 0;
for (k = 0; k < 5; k++)
{
    for (i = 0; i < n; i++)
    {
        if (f[i] == 0)
        {
            int flag = 0;
            for (j = 0; j < m; j++)
            {
                if (need[i][j] > avail[j])
                {
                    flag = 1;
                    break;
                }
            }
            if (flag == 0)
            {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
```

```
                f[i] = 1;
            }
        }
    }
}
int flag = 1;
for (int i = 0; i < n; i++)
{
    if (f[i] == 0)
    {
        flag = 0;
        printf("The following system is not safe");
        break;
    }
}
if (flag == 1)
{
    printf("Following is the SAFE Sequence\n");
    for (i = 0; i < n - 1; i++)
        printf(" P%d ->", ans[i]);
    printf(" P%d", ans[n - 1]);
}
return (0);
}
```

Output:-

```
Following is the SAFE Sequence
 P1 -> P3 -> P4 -> P0 -> P2
```

Experiment -6

Write a C program to implement DFA for the given regular expression and test whether the given string is accepted or not

```
#include <stdio.h>
#include <strings.h> // Note: Consider using <string.h> instead
#include <stdlib.h> // For exit() function

void main() {
    int table[2][2], i, j, l, status = 0;
    char input[100];
```

```
    printf("To implement DFA of language (a+aa*b)* \nEnter Input String:");

    // Define DFA transition table
    table[0][0] = 1; // state 0 on 'a' goes to state 1
    table[0][1] = -1; // state 0 on 'b' goes to -1 (invalid)
    table[1][0] = 1; // state 1 on 'a' goes to state 1
    table[1][1] = 0; // state 1 on 'b' goes to state 0

    scanf("%s", input);
    l = strlen(input);

    // Check each character in the input string
    for (i = 0; i < l; i++) {
        if (input[i] != 'a' && input[i] != 'b') {
            printf("\nThe entered Value is wrong");
            getch(); // Note: getch() is non-standard, consider removing
            exit(0);
        }

        // Transition based on current character
        if (input[i] == 'a') {
            status = table[status][0];
        } else {
            status = table[status][1];
        }

        // If at any point, the DFA reaches state -1, print string not accepted
        if (status == -1) {
            printf("String not Accepted");
            break;
        }
    }

    // If end of string is reached and status is not -1, string is accepted
    if (i == l) {
        printf("String Accepted");
    }
}
```

Output:
Run 1:

To implementing DFA of language (a+aa*b)*
Enter Input String:cbsd
The entered Value is wrong.
Run 2:
To implementing DFA of language (a+aa*b)*
Enter Input String:abbababa
String not Accepted.
Run 3:
To implementing DFA of language (a+aa*b)*
Enter Input String:babbaab
String not Accepted.

Experiment -7

Write a program to construct NFA from the given regular expression and test whether the given string is accepted or not.

PROGRAM

```
#include <stdio.h>
#include <string.h>
void main()
{
char str[100];
char state='P';
int i=0;
printf("Enter input string: \n");
scanf("%s",str);
while(str[i]!='\0')
{
switch(state)
{
case 'P':
if (str[i]=='a') state='Q';
else if (str[i]=='b') state='P';
break;
case 'Q':
if (str[i]=='b') state='R';
else state='T';
break;
case 'R':
if (str[i]=='b') state='S';
else state='T';
break;
case 'S':
if (str[i]=='b') state='P';
```

```
else state='T';
break;
case 'T':
if (str[i]=='b') state='R';
else state='T';
break;
}
i++;
}
if (state=='S')
{
printf("String Accepted\n");
}
else
{
printf("String not Accepted\n");
}
}
```

OUTPUT

```
Enter input string:
(a+aa*b)*
String Accepted
```

Experiment -8

Write a C program to identify different types of Tokens in a given Program.

```
#include<ctype.h>
#include<stdio.h>
#include<string.h>
main()
{
int i=0,f,k=0,j,l,n,a,a1;
char temp[10];
char s[100],g[100];
printf( "Enter Program $ for termination:\n");
do
{
gets(g);
if(strcmp(g,"$")==0) goto s1;
for(a1=0;g[a1]!='\0';a1++,i++)
s[i]=g[a1];
```

```
}
while(1);
s1:
s[i]='\0';
i=0;
printf("\nvariables:");
while(s[i]!='\0')
{
if(isalpha(s[i]))
{
j=i;
while(isalnum(s[i+1])||s[i+1]=='['||s[i+1]==']')
{
i++;
}
if(s[i+1]==' '||s[i+1]=='('||s[i+1]=='{'||s[i+1]=='\n')
{
i++;
}
else
{
for(;j<=i;j++)
printf("%c",s[j]);
}
printf("");
}
i++;
} /*end of while*/
i=0;
printf("\nOperators:");
while(s[i]!='\0')
{
if(s[i]=='='||s[i]=='+'||s[i]=='-'||s[i]=='*'||s[i]=='/'||s[i]=='>'||s[i]=='<')
{
printf("%c",s[i]);
printf("");
}
i++;
} /* end of while */
i=0;
```

```
printf("\nconstants:");
while(s[i]!='\0')
{
if(isalpha(s[i]))
{
while(isalnum(s[i+1])||s[i+1]=='['||s[i+1]==']')
i++;
i++;
}
if(isdigit(s[i]))
{
k=i;
while(isdigit(s[i+1]))
{
i++;
}
for(;k<=i;k++)
printf("%c",s[k]);
printf("");
} /*end of if (after while)*/
i++;
} /*end of while*/
i=0;
printf("\nspecial symbols:");
while(s[i]!='\0')
{
if(s[i]==';'||s[i]==','||s[i]=='('||s[i]==')'||s[i]=='{'||s[i]=='}'||s[i]=='['||s[i]==']')
printf("%c",s[i]);
i++;
}
i=0;
printf("\nkeywords:");
while(s[i]!='\0')
{
if(isalpha(s[i]))
{
j=i;
while(isalpha(s[i+1]))
{
i++;
```

```
}
if(s[i+1]==' ')
{
for(;j<=i;j++)
printf("%c",s[j]);
}
else
{
printf("");
}
}
i++;
} /*end of while*/
}
```

OUTPUT:-

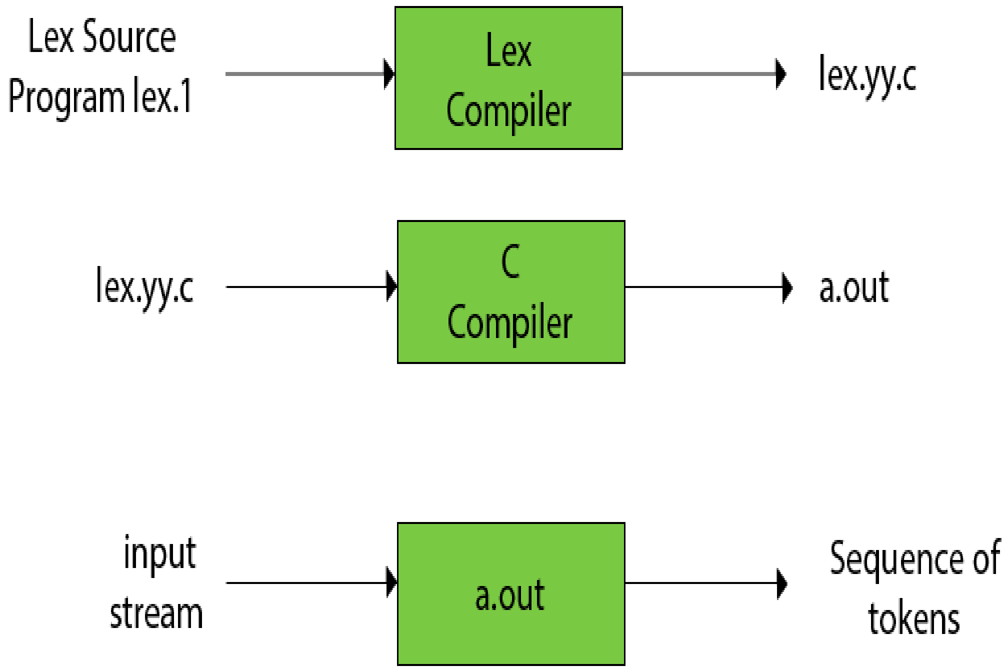
```
main()
{
int a,b,c;
c=245;
a=b+c;
}
```

Experiment -9

Write a Lex Program to implement a Lexical Analyzer using Lex tool

The function of Lex is as follows:

- Firstly lexical analyzer creates a program lex.1 in the Lex language. Then Lex compiler runs the lex.1 program and produces a C program lex.yy.c.
- Finally C compiler runs the lex.yy.c program and produces an object program a.out.
- a.out is lexical analyzer that transforms an input stream into a sequence of tokens.



Lex file format

A Lex program is separated into three sections by %% delimiters. The formal of Lex source is as follows:

1.{ definitions }

2.%%

3. { rules }

4.%%

5.{ user subroutines }

Definitions include declarations of constant, variable and regular definitions.

Rules define the statement of form p1 {action1} p2 {action2}....pn {action}.

```
%{
#include <stdio.h>
%}

 %%
[0-9]+ { printf("Saw an integer: %s\n", yytext); }
[a-zA-Z]+ { printf("Saw an String: %s\n", yytext); }
```

```
%%

main( )
{
printf("Enter some input \n");
yylex();
}

int yywrap()
{
return 1;
}
```

Program -2

```
%{
#include<stdio.h>
#include<string.h>
int i = 0;
%}

/* Rules Section*/
%%
([a-zA-Z0-9])*    {i++;} /* Rule for counting
                         number of words*/

"\n" {printf("%d No of words\n", i); i = 0;}
%%

int yywrap(void){}

int main()
{
   // The function that starts the analysis
   yylex();

   return 0;
}
```

Experiment -10

Write a parsing program to test whether the given expression is having balanced parenthesis or not

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

```
#define MAX 100

// Stack structure
typedef struct {
    char items[MAX];
    int top;
} Stack;

// Function to initialize the stack
void initStack(Stack *s) {
    s->top = -1;
}

// Function to check if the stack is empty
bool isEmpty(Stack *s) {
    return s->top == -1;
}

// Function to push an element onto the stack
void push(Stack *s, char c) {
    if (s->top < MAX - 1) {
        s->items[++(s->top)] = c;
    } else {
        printf("Stack overflow\n");
        exit(1);
    }
}

// Function to pop an element from the stack
char pop(Stack *s) {
    if (isEmpty(s)) {
        printf("Stack underflow\n");
        exit(1);
    }
    return s->items[(s->top)--];
}

// Function to check the top element of the stack
char peek(Stack *s) {
```

```
    if (isEmpty(s)) {
        return '\0';  // Return a null character if stack is empty
    }
    return s->items[s->top];
}

// Function to check if the parentheses are balanced
bool areBalanced(char *expression) {
    Stack stack;
    initStack(&stack);
int i;
    for (i = 0; expression[i] != '\0'; i++) {
        char c = expression[i];

        // Push opening parentheses onto the stack
        if (c == '(' || c == '{' || c == '[') {
            push(&stack, c);
        }
        // Check closing parentheses
        else if (c == ')' || c == '}' || c == ']') {
            if (isEmpty(&stack)) {
                return false;  // No matching opening parenthesis
            }

            char top = pop(&stack);
            if ((c == ')' && top != '(') ||
                (c == '}' && top != '{') ||
                (c == ']' && top != '[')) {
                return false;  // Mismatched parentheses
            }
        }
    }

    // If stack is empty, parentheses are balanced
    return isEmpty(&stack);
}

int main() {
    char expression[MAX];
```

```
    printf("Enter an expression: ");
    fgets(expression, sizeof(expression), stdin);

    // Remove newline character if present
    size_t length = strlen(expression);
    if (length > 0 && expression[length - 1] == '\n') {
        expression[length - 1] = '\0';
    }

    if (areBalanced(expression)) {
        printf("The expression has balanced parentheses.\n");
    } else {
        printf("The expression does not have balanced parentheses.\n");
    }

    return 0;
}
```

Experiment -11

Write a C program for implementation of a Shift Reduce Parser using Stack Data Structure to accept a given input string of a given grammar

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
//Global Variables
int z = 0, i = 0, j = 0, c = 0;
// Modify array size to increase
// length of string to be parsed
char a[16], ac[20], stk[15], act[10];
// This Function will check whether
// the stack contain a production rule
// which is to be Reduce.
// Rules can be E->2E2 , E->3E3 , E->4
void check()
{
// Copying string to be printed as action
strcpy(ac,"REDUCE TO E -> ");
// c=length of input string
for(z = 0; z < c; z++)
```

```
{
//checking for producing rule E->4
if(stk[z] == '4')
{
printf("%s4", ac);
stk[z] = 'E';
stk[z + 1] = '\0';
//printing action
printf("\n$%s\t%s$\t", stk, a);
}
}
for(z = 0; z < c - 2; z++)
{
//checking for another production
if(stk[z] == '2' && stk[z + 1] == 'E' &&
stk[z + 2] == '2')
{
printf("%s2E2", ac);
stk[z] = 'E';
stk[z + 1] = '\0';
stk[z + 2] = '\0';
printf("\n$%s\t%s$\t", stk, a);
i = i - 2;
}
}
for(z=0; z<c-2; z++)
{
//checking for E->3E3
if(stk[z] == '3' && stk[z + 1] == 'E' &&
stk[z + 2] == '3')
{
printf("%s3E3", ac);
stk[z]='E';
stk[z + 1]='\0';
stk[z + 1]='\0';
printf("\n$%s\t%s$\t", stk, a);
i = i - 2;
}
}
return ; //return to main
```

```
}
//Driver Function
int main()
{
printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
// a is input string
strcpy(a,"32423");
// strlen(a) will return the length of a to c
c=strlen(a);
// "SHIFT" is copied to act to be printed
strcpy(act,"SHIFT");
// This will print Labels (column name)
printf("\nstack \t input \t action");
// This will print the initial
// values of stack and input
printf("\n$\t%s$\t", a);
// This will Run upto length of input string
for(i = 0; j < c; i++, j++)
{
// Printing action
printf("%s", act);
// Pushing into stack
stk[i] = a[j];
stk[i + 1] = '\0';
// Moving the pointer
a[j]=' ';
// Printing action
printf("\n$%s\t%s$\t", stk, a);
// Call check function ..which will
// check the stack whether its contain
// any production or not
check();
}
// Rechecking last time if contain
// any valid production then it will
// replace otherwise invalid
check();
// if top of the stack is E(starting symbol)
// then it will accept the input
if(stk[0] == 'E' && stk[1] == '\0')
```

```
printf("Accept\n");
else //else reject
printf("Reject\n");
}
```

Experiment -12

Write a C program to implement a Recursive Descent Parser

```
#include<stdio.h>
#include<string.h>
int E(),Edash(),T(),Tdash(),F();
char *ip;
char string[50];
int main()
{
printf("Enter the string\n");
scanf("%s",string);
ip=string;
printf("\n\nInput\tAction\n--------------------------------\n");
if(E() && ip=="\0"){
printf("\n--------------------------------\n");
printf("\n String is successfully parsed\n");
}
else{
printf("\n--------------------------------\n");
printf("Error in parsing String\n");
}
}
int E()
{
printf("%s\tE->TE' \n",ip);
if(T())
{
if(Edash())
{
return 1;
}
else
return 0;

}
```

```
else
return 0;
}
int Edash()
{
if(*ip=='+')
{
printf("%s\tE'->+TE' \n",ip);
ip++;
if(T())
{
if(Edash())
{
return 1;
}
else
return 0;
}
else
return 0;
}
else
{
printf("%s\tE'->^ \n",ip);
return 1;
}
}
int T()
{
printf("%s\tT->FT' \n",ip);
if(F())
{
if(Tdash())
{

return 1;
}
else
return 0;
}
```

```
else
return 0;
}
int Tdash()
{
if(*ip=='*')
{
printf("%s\tT'->*FT' \n",ip);
ip++;
if(F())
{
if(Tdash())
{
return 1;
}
else
return 0;
}
else
return 0;
}
else
{
printf("%s\tT'->^ \n",ip);
return 1;
}
}
int F()
{
if(*ip=='(')

{
printf("%s\tF->(E) \n",ip);
ip++;
if(E())
{
if(*ip==')')
{
ip++;
return 0;
```

```
}
else
return 0;
}
else
return 0;
}
else if(*ip=='i')
{
ip++;
printf("%s\tF->id \n",ip);
return 1;
}
else
return 0;
}
```

Experiment -13

13a) Write a program to determine FIRST sets for all variables and terminals from the given CFG.

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void addToResultSet(char[],char);
int numOfProductions;
char productionSet[10][10];
main()
{
int i;
char choice;
char c;
char result[20];
printf("How many number of productions ? :");
scanf(" %d",&numOfProductions);
for(i=0;i<numOfProductions;i++)//read production string eg: E=E+T
{
printf("Enter productions Number %d : ",i+1);
scanf(" %s",productionSet[i]);
}
```

```
do
{
printf("\n Find the FIRST of :");
scanf(" %c",&c);
FIRST(result,c); //Compute FIRST; Get Answer in 'result' array
printf("\n FIRST(%c)= { ",c);
for(i=0;result[i]!='\0';i++)
printf(" %c ",result[i]); //Display result
printf("}\n");
printf("press 'y' to continue : ");
scanf(" %c",&choice);
}

while(choice=='y'||choice =='Y');
}
void FIRST(char* Result,char c)
{
int i,j,k;
char subResult[20];
int foundEpsilon;
subResult[0]='\0';
Result[0]='\0';
if(!(isupper(c)))
{
addToResultSet(Result,c);
return ;
}
for(i=0;i<numOfProductions;i++)
{
if(productionSet[i][0]==c)
{
if(productionSet[i][2]=='$') addToResultSet(Result,'$');
else
{
j=2;
while(productionSet[i][j]!='\0')
{
foundEpsilon=0;
FIRST(subResult,productionSet[i][j]);
for(k=0;subResult[k]!='\0';k++)
```

```
addToResultSet(Result,subResult[k]);
for(k=0;subResult[k]!='\0';k++)
if(subResult[k]=='$')
{
foundEpsilon=1;
break;
}
if(!foundEpsilon)
break;
j++;
}
}
}
}
return ;
}
void addToResultSet(char Result[],char val)
{
int k;
for(k=0 ;Result[k]!='\0';k++)
if(Result[k]==val)
return;
Result[k]=val;
Result[k+1]='\0';
}
```

OUTPUT

```
How many number of productions ? :4
Enter productions Number 1 : E=TR
Enter productions Number 2 : R=+TR
Enter productions Number 3 : T=a
Enter productions Number 4 : Y=s
Find the FIRST of :E
FIRST(E)= { a }
press 'y' to continue : y
Find the FIRST of :R
FIRST(R)= { + }
press 'y' to continue : y
Find the FIRST of :Y
FIRST(Y)= { s }
```

press 'y' to continue :

13 b) Write a program to determine FOLLOW sets for all variables from the given CFG.

```
#include<stdio.h>
#include<string.h>
int n,m=0,p,i=0,j=0;
char a[10][10],followResult[10];
void follow(char c);
void first(char c);
void addToResult(char);
int main()
{
int i;
int choice;
char c,ch;
printf("Enter the no. of productions: ");
scanf("%d", &n);
printf(" Enter %d productions\n Production with multiple terms should be give as separate productions \n", n);
for(i=0;i<n;i++)
scanf("%s%c",a[i],&ch);
// gets(a[i]);
do
{
m=0;
printf("Find FOLLOW of -->");
scanf(" %c",&c);
follow(c);
printf("FOLLOW(%c) = { ",c);
for(i=0;i<m;i++)
printf("%c ",followResult[i]);
printf(" }\n");
printf("Do you want to continue(Press 1 to continue....)?");
scanf("%d%c",&choice,&ch);
}
while(choice==1);
}
void follow(char c)
{
if(a[0][0]==c)addToResult('$');
```

```
for(i=0;i<n;i++)
{
for(j=2;j<strlen(a[i]);j++)
{
if(a[i][j]==c)
{
if(a[i][j+1]!='\0')first(a[i][j+1]);
if(a[i][j+1]=='\0'&&c!=a[i][0])
follow(a[i][0]);
}
}
}
}
void first(char c)
{
int k;
if(!(isupper(c)))
//f[m++]=c;
addToResult(c);
for(k=0;k<n;k++)
{
if(a[k][0]==c)
{
if(a[k][2]=='$') follow(a[i][0]);

else if(islower(a[k][2]))
//f[m++]=a[k][2];
addToResult(a[k][2]);
else first(a[k][2]);
}
}
}
void addToResult(char c)
{
int i;
for( i=0;i<=m;i++)
if(followResult[i]==c)
return;
followResult[m++]=c;
}
```

OUTPUT

```
Enter the no. of productions: 6
Enter 6 productions
Production with multiple terms should be give as separate productions
E=TR
R=+TR
T=FY
Y=*FY
F=(E)
F=a
Find FOLLOW of -->E
FOLLOW(E) = { $ ) }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->R
FOLLOW(R) = { ) }
Do you want to continue(Press 1 to continue....)?
```

Experiment -14

14**) Write a program which takes predictive parsing table as input and to determine whether the input string is accepted or not.**

PROGRAM:

```
#include <stdio.h>
#include <string.h>
char prol[7][10] = {"S", "A", "A", "B", "B", "C", "C"};
char pror[7][10] = {"A", "Bb", "Cd", "aB", "@", "Cc", "@"};
char prod[7][10] = {"S->A", "A->Bb", "A->Cd", "B->aB", "B->@", "C->Cc", "C->@"};
char first[7][10] = {"abcd", "ab", "cd", "a@", "@", "c@", "@"};
char follow[7][10] = {"$", "$", "$", "a$", "b$", "c$", "d$"};
char table[5][6][10];
int numr(char c) {
switch (c) {
case 'S': return 0;
case 'A': return 1;
case 'B': return 2;
case 'C': return 3;
case 'a': return 0;
case 'b': return 1;
```

```
case 'c': return 2;
case 'd': return 3;
case '$': return 4;
}
return 2;
}
int main() {
int i, j, k;
clrscr();
for (i = 0; i < 5; i++)
for (j = 0; j < 6; j++)
strcpy(table[i][j], " ");
```

```
printf("\nThe following is the predictive parsing table for the following grammar:\n");
for (i = 0; i < 7; i++)
printf("%s\n", prod[i]);
printf("\nPredictive parsing table is\n");
for (i = 0; i < 7; i++) {
k = strlen(first[i]);
for (j = 0; j < k; j++)
if (first[i][j] != '@')
strcpy(table[numr(prol[i][0]) + 1][numr(first[i][j]) + 1], prod[i]);
}
for (i = 0; i < 7; i++) {
if (strlen(pror[i]) == 1) {
if (pror[i][0] == '@') {
k = strlen(follow[i]);
for (j = 0; j < k; j++)
strcpy(table[numr(prol[i][0]) + 1][numr(follow[i][j]) + 1], prod[i]);
}
}
}
strcpy(table[0][0], " ");
strcpy(table[0][1], "a");
strcpy(table[0][2], "b");
strcpy(table[0][3], "c");
strcpy(table[0][4], "d");
strcpy(table[0][5], "$");
strcpy(table[1][0], "S");
strcpy(table[2][0], "A");
strcpy(table[3][0], "B");
strcpy(table[4][0], "C");
printf("\n--------------------------------------------------------\n");
for (i = 0; i < 5; i++)
```

```
for (j = 0; j < 6; j++) {
printf("%-10s", table[i][j]);
if (j == 5)
printf("\n--------------------------------------------------------\n");
}
getchar();
return 0;
}
```

INPUT & OUTPUT:

The following is the predictive parsing table for the following grammar:

```
S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@
Predictive parsing table is
-----------------------------------------------------------------
a b c d $
-----------------------------------------------------------------
S S->AS->AS->AS->A
----------------------------------------------------------------
A A->Bb A->BbA->Cd A->Cd
-----------------------------------------------------------------
B B->aB B->@ B->@ B->@
-----------------------------------------------------------------
C C->@C->@ C->@
-----------------------------------------------------------------
```

<u>Experiment -15</u>

Simulate the calculator using LEX and YACC tool.

<u>INSTALLATION:-</u>

1. sudo apt-get update
2. sudo apt-get install flex
3. sudo apt-get install bison

4.sudo apt-get install byacc

5.sudo apt-get install bison++

6.sudo apt-get install byacc –j

Create LEX File (calc.l)

```
%{
#include<stdio.h>
#include "y.tab.h"
extern int yylval;
%}

%%
[0-9]+ {
        yylval=atoi(yytext);
        return NUMBER;
      }
[\t] ;
[\n] return 0;
. return yytext[0];
%%

int yywrap()
{
return 1;
}
```

Create YACC File (`calc.y`)

```
%{
   #include<stdio.h>
   int flag=0;
  %}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{
         printf("\nResult=%d\n",$$);
         return 0;
        };
E:E'+'E {$$=$1+$3;}
 |E'-'E {$$=$1-$3;}
 |E'*'E {$$=$1*$3;}
 |E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}
 |'('E')' {$$=$2;}
 | NUMBER {$$=$1;}
;
%%
```

```
void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

   yyparse();
  if(flag==0)
   printf("\nEntered arithmetic expression is Valid\n\n");

}

void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
}
```

Generate the LEX C code:
lex calc.l

This command generates lex.yy.c.
Generate the YACC C code:
bison -d calc.y (**or)** yacc –d cal.y
This command generates y.tab.c and y.tab.h. The -d flag creates the header file y.tab.h.

Compile the generated C files:
gcc y.tab.c lex.yy.c -ll –ly

Run the calculator:
./a.out