1. Given the list of processes, their burst times and arrival times, Write a C program to implement the FCFS CPU scheduling algorithm. Display the turnaround time & waiting time for each process. Also calculate the average turnaround time and average waiting time.

```
#include<stdio.h>
#include<stdlib.h>
int n,a[100],b[100],v[100],start[100],stop[100];
void gantt()
{
	int min,i,j,ind,dum=0;
	for(j=0;j<n;j++)
	{
		min=999;
		for(i=0;i<n;i++)
		{
			if(!v[i] && a[i]<min)
			{
				min=a[i];
				ind=i;
			}
		}
		v[ind]=1;
		if(dum<a[ind])
		{
			start[ind]=a[ind];
		}
		else
		{
			start[ind]=dum;
		}
		stop[ind]=start[ind]+b[ind];
		dum=stop[ind];
	}
}
void avgwait()
{
	int wait[100],i;
	float avg=0;
	printf("The waiting times :\n");
	for(i=0;i<n;i++)
	{
		wait[i]=start[i]-a[i];
		avg+=wait[i];
		printf("P%d : %d ms\n",i+1,wait[i]);
	}
	avg=avg/n;
	printf("Average waiting time = %f\n",avg);
}
void avgtat()
{
	int tat[100],i;
	float avg=0;
	printf("The turn around times :\n");
	for(i=0;i<n;i++)
	{
		tat[i]=stop[i]-a[i];
		avg+=tat[i];
		printf("P%d : %d ms\n",i+1,tat[i]);
	}
	avg=avg/n;
	printf("Average turn around time = %f\n",avg);
}
int main()
{
```

```
	int i;
	printf("Enter the no. of processes : ");
	scanf("%d",&n);
	printf("Enter the arrival time :\n");
	for(i=0;i<n;i++)
	{
		printf("P%d : ",i+1);
		scanf("%d",&a[i]);
		v[i]=0;
		start[i]=stop[i]=0;
	}
	printf("Enter the burst time :\n");
	for(i=0;i<n;i++)
	{
		printf("P%d : ",i+1);
		scanf("%d",&b[i]);
	}
	gantt();
	avgwait();
	avgtat();
	return 0;
}
```

OUTPUT :

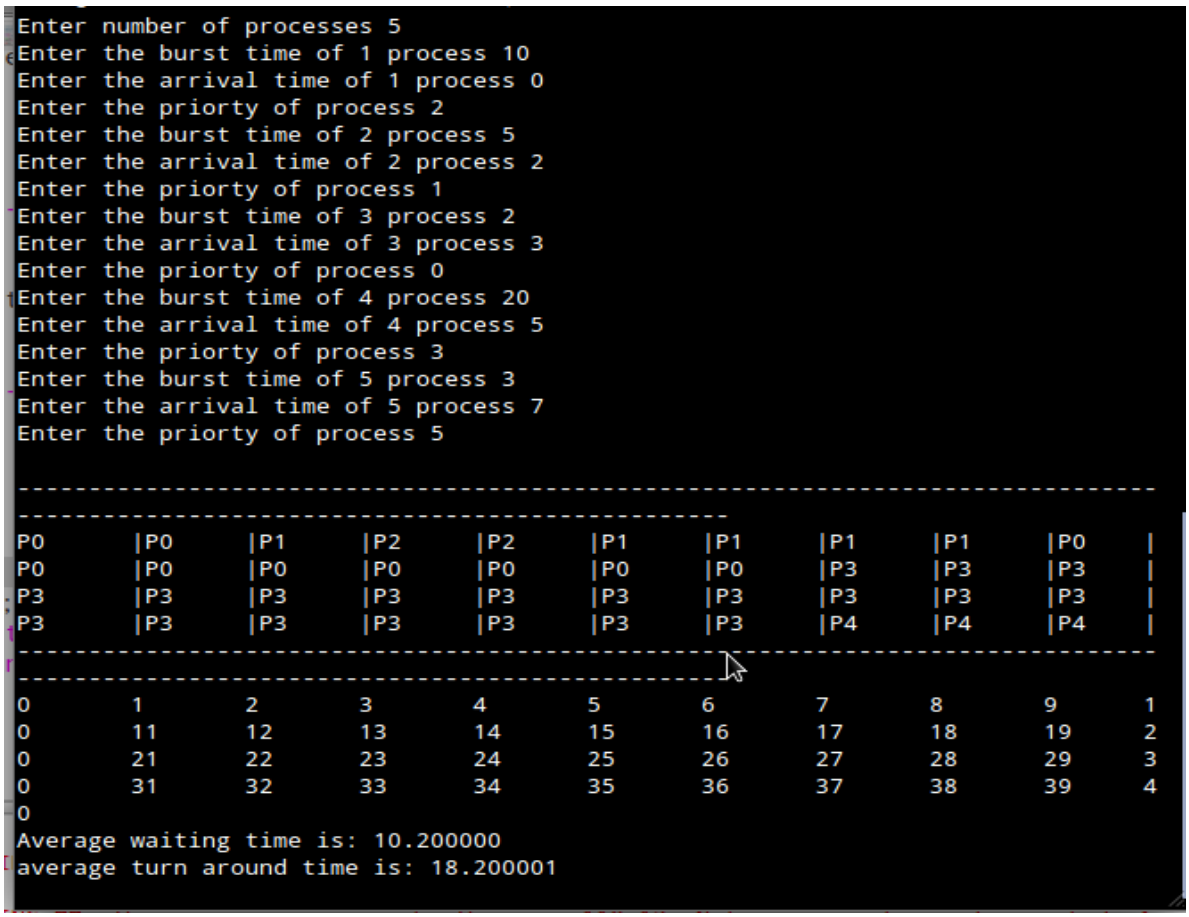
```
user@dell-version-2-8:~/1SI17CS412$ gcc FCFS.c
user@dell-version-2-8:~/1SI17CS412$ ./a.out
Enter the no. of processes : 3
Enter the arrival time :
P1 : 0
P2 : 1
P3 : 2
Enter the burst time :
P1 : 24
P2 : 3
P3 : 3
The waiting times :
P1 : 0 ms
P2 : 23 ms
P3 : 25 ms
Average waiting time = 16.000000
The turn around times :
P1 : 24 ms
P2 : 26 ms
P3 : 28 ms
Average turn around time = 26.000000
user@dell-version-2-8:~/1SI17CS412$
```

2. Given the list of processes, their burst times, priority and arrival times, Write a C program to implement the preemptive priority CPU scheduling algorithm. Display the turnaround time & waiting time for each process. Also calculate the average turnaround time and average waiting time.

```
 #include<stdio.h>
#include<stdlib.h>
struct time
{
	int bt,at,tat,wt,rt,pt,rtime;
	//int p;
};
int main()
{
```

```
	struct time p[10];
	int i,j,k,n,ttat=0,twt=0;
	int endtime,gant[100];
	float awt,atat;
	int remain=0,t;
	int min,process;
	printf("Enter number of processes");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter the burst time of %d process",i+1);
		scanf("%d",&p[i].bt);
		printf("Enter the arrival time of %d process",i+1);
		scanf("%d",&p[i].at);
		printf("Enter the priorty of process");
		scanf("%d",&p[i].pt);
		//p[i].p=i;
		p[i].rtime=p[i].bt;
	}
	for(t=0;remain!=n;t++)
	{
		min=999;
		for(i=0;i<n;i++)
		{
			if(p[i].at<=t&&p[i].pt<min&&p[i].rtime>0)
			{
				gant[t]=i;
				min=p[i].pt;
				process=i;
			}
		}
		p[process].rtime--;
		if(p[process].rtime==0)
		{
			remain++;
			endtime=t+1;
			p[process].wt=endtime-p[process].bt-p[process].at;
			p[process].tat=endtime-p[process].at;
			ttat+=endtime-p[process].at;
			twt+=endtime-p[process].bt-p[process].at;
		}
	}
	printf("\n");
	for(i=0;i<2*n;i++)
	printf("----------");
	printf("\n");
	for(i=0;i<t;i++)
	printf("P%d\t|",gant[i]);
	printf("\n");
	for(i=0;i<2*n;i++)
	printf("----------");
	printf("\n");
	printf("0\t");
	for(i=1;i<t+1;i++)
	printf("%d\t",i);
	printf("\n");
	awt=(float)twt/n;
	atat=(float)ttat/n;
	printf("Average waiting time is: %f\n",awt);
	printf("average turn around time is: %f\n",atat);

	printf("\n");
	return 0;
}
```

Output :

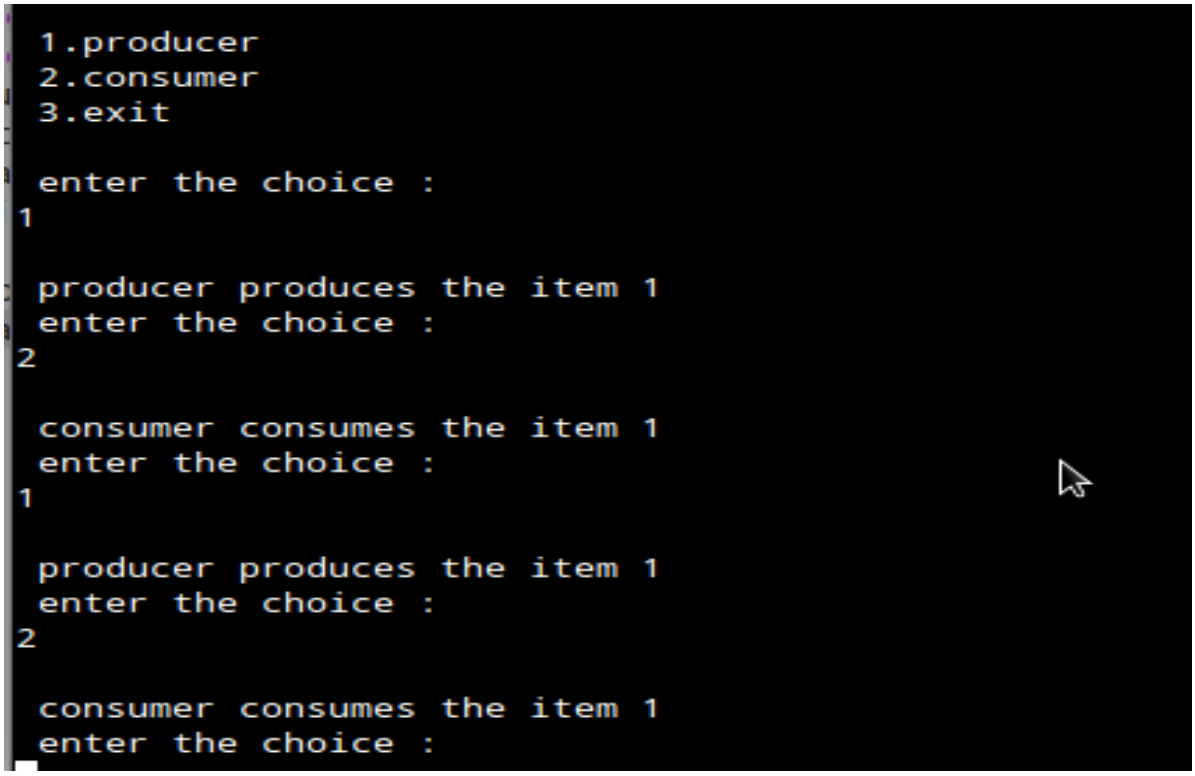


3. Write a C program to implement producer-consumer problem using semaphores.

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
void consumer();
int wait(int);
int signal(int);

void producer();
int main()
{
	int ch;
	printf("\n 1.producer\n 2.consumer\n 3.exit\n");
	while(1)
	{

		printf("\n enter the choice :\n");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1: if((mutex==1)&&(empty!=0))
					producer();
				else
					printf("\n buffer is full\n");
				break;
			case 2: if((mutex==1)&&(full!=0))
					consumer();
				else
					printf("buffer is empty \n");
				break;
			case 3:exit(0);
				break;
```

```
            }
        }
        return 0;
}
int wait(int s)
{
        return(--s);
}
int signal(int s)
{
        return(++s);
}
void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\n producer produces the item %d",x);
        mutex=signal(mutex);
}
void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        x--;
        printf("\n consumer consumes the item %d",x+1);
        mutex=signal(mutex);
}
```

Output :



4. Write a C program to implement Bankers algorithm for the purpose of deadlock avoidance.

```
 #include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#define SIZE 100
int N,R;
```

```
int allocation[SIZE][SIZE],max[SIZE][SIZE],need[SIZE][SIZE],available[SIZE];
bool finished[SIZE];
int safeSequence[SIZE];
void getInputData()
{
   int i=0,j=0 ;
   printf("Enter number of processes : ");
   scanf("%d", &N);
   printf("Enter number of Resource types : ");
   scanf("%d", &R);
   printf("Enter Allocation matrix : \n");
   for (i = 0; i < N; i++)
   {
      for (j = 0; j < R; j++)
      {
         scanf("%d", &allocation[i][j]);
      }
   }
   printf("Enter Max matrix : \n");
   for (i = 0; i < N; i++)
   {
      for (j = 0; j < R; j++)
      {
         scanf("%d", &max[i][j]);
         need[i][j] = max[i][j] - allocation[i][j];
      }
   }
          printf("\n Need matrix: \n");
          for(i=0;i<N;i++)
          {
                  for(j=0;j<R;j++)
                          printf("%d\n",need[i][j]);
                          printf("\n");
          }
   printf("Enter Available array : \n");
   for (i = 0; i < R; i++)
      scanf("%d", &available[i]);
}
int getNextProcess(int work[], int current)
{
   int i =0,j = 0 ;
   for (i = 0 ; i < N ; i++  )
   {
      current = (current+1)%N ;
      if(finished[current]) continue ;
      bool flag = true;
      for (j = 0; j < N; j++)
      {
         if (need[current][j] > work[j])
         {
            flag = false;
            break;
         }
      }
      if (flag == false)
         continue;
      return current;
   }
   return -1;
}
bool findSafeSequence()
{
   int *work = available;
   int safeSequencePointer = 0;
```

```
    int current = -1 , i =0 ;
    while (true)
    {
        int next = getNextProcess(work, current);
        if (next < 0)
        {
            for (i = 0; i < N; i++)
                if (finished[i] == false)
                    return false;
            return true;
        }
        current = next;
        finished[next] = true;
        safeSequence[safeSequencePointer++] = next;
        for (i = 0; i < R; i++)
        {
            work[i] += allocation[next][i];
        }
    }
}
void displaySafeSequence()
{
    int i =0 ;
    printf("\nSafe Sequence is : \n");
    for (i = 0; i < N; i++)
    {
        printf("P%d ", safeSequence[i]);
    }
    printf("\n");
}
int main()
{
    getInputData();
    if (findSafeSequence() == true)
    {
        displaySafeSequence();
    }
    else
    {
        printf("\nSo safe sequence exists !");
    }
}
```

OR

```
 #include<stdio.h>
#include<stdlib.h>
int main()
{
            int allocation[10][5],need[10][5],MAX[10][5],available[10];
            int i,j,n,r,arr[10],flag[10],remain,k=0,dead;
            printf("Enter  number of processes and resources\n");
            scanf("%d%d",&n,&r);
            remain=n;
            dead=n*n;
            printf("Enter the allocation matrix\n");
            for(i=0;i<n;i++)
            {
                        flag[i]=0;
                        for(j=0;j<r;j++)
                        scanf("%d",&allocation[i][j]);
            }
            printf("Enter MAX Matrix\n");
            for(i=0;i<n;i++)
```

```
        for(j=0;j<r;j++)
        {
                scanf("%d",&MAX[i][j]);
                need[i][j]=MAX[i][j]-allocation[i][j];
        }
        printf("Enter available resources:\n");
        for(j=0;j<r;j++)
        scanf("%d",&available[j]);
        for(i=0;remain!=0;)
        {
                j=0;
                if(flag[i]==0)
                {
                        while(flag[i]==0&&need[i][j]<=available[j])
                        {
                                j++;
                                if(j==r)
                                flag[i]=1;
                        }
                        if(flag[i]==1)
                        {
                                for(j=0;j<r;j++)
                                available[j]+=allocation[i][j];
                                arr[k++]=i;
                                remain--;
                        }
                }
                i=(i+1)%n;
                if((dead--)==0)
                break;
        }
        if(remain==0)
        {
                printf("\n\nSafe sequence exists\nthe safe sequence is\n");
                for(i=0;i<n;i++)
                printf("P%d\n",arr[i]);
                printf("\n");
        }
        else
        printf("\n\n System is in deadlock state :\n");
        return 0;
}
```

Output :

```
Enter number of processes : 5
Enter number of Resource types : 3
Enter Allocation matrix :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter Max matrix :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

 Need matrix:
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
Enter Available array :
3 3 2

Safe Sequence is :
P1 P3 P4 P0 P2 user@dell-version-2-8:~/1SI17CS412$
```

5. Write a C program to implement the following contiguous memory allocation techniques : a) Worst-fit
b) Best-fit
c) First-fit

```
#include <stdio.h>
#include <stdlib.h>
int processMemory[100], tempMemory[100],  memory[100];
int noMemoryBlock, noProcesses;
void fnFirstFit(int memory[])
{
   int i,j;
   printf("\nFirstFit\nProcess\t\tMemoryBlock");
   for(i=0;i<noProcesses;i++)
   {
      int flag=0;
      for(j=0;j<noMemoryBlock;j++)
         if(processMemory[i] <= memory[j])
         {
            flag=1;
            memory[j]-=processMemory[i];
            printf("\n%d\t\t%d",i+1,j+1);
            break;
         }
      if(flag==0)
         return;
   }
}

void fnWorstFit(int memory[100])
{
   int i,j;
   printf("\nWorstFit\nProcess\t\tMemoryBlock");
   for(i=0;i<noProcesses;i++)
   {
      int high=-1;
      for(j=0;j<noMemoryBlock;j++)
         if(processMemory[i]<=memory[j])
            if(memory[high]<memory[j] || high == -1)
               high = j;
      if(high != -1)
      {
         memory[high]-=processMemory[i];
         printf("\n%d\t\t%d",i+1,high+1);
      }
      else
      {
         printf("Cant allocate further");
         return;
      }
   }
}
void fnBestFit(int memory[100])
{
   int i,j;
   printf("\nBestFit\nProcess\t\tMemoryBlock");
   for(i=0;i<noProcesses;i++)
   {
      int low=-1;
      for(j=0;j<noMemoryBlock;j++)
         if(processMemory[i]<=memory[j])
            if(memory[low]>memory[j] || low == -1)
               low = j;
      if(low != -1)
```

```
        {
            memory[low]-=processMemory[i];
            printf("\n%d\t\t%d",i+1,low+1);
        }
        else
        {
            printf("Cant allocate further");
            return;
        }
    }
}
void restore()
{
    int i;
    for(i=0;i<noMemoryBlock;i++)
        memory[i]=tempMemory[i];
}
int main()
{
    int i, choice;
    printf("\nEnter the total number of memory blocks and number requested processes:");
    scanf("%d%d",&noMemoryBlock, &noProcesses);

    printf("\nEnter the size of memory block:\n");
    for(i=0;i<noMemoryBlock;i++)
        scanf("%d",&tempMemory[i]);

    printf("\nEnter the size of memory requested by process:\n");
    for(i=0;i<noProcesses;i++)
        scanf("%d",&processMemory[i]);

    while(1)
    {
        printf("\nEnter 1:FirstFit, 2.BestFit, 3. WorstFit\nEnter your choice:");
        scanf("%d",&choice);
        restore();
        if(choice == 1)     fnFirstFit(memory);
        else if(choice == 2)     fnBestFit(memory);
        else if(choice == 3)     fnWorstFit(memory);
        else
          exit(0);
    }
}
```

OUTPUT :

```
Enter the total number of memory blocks and number requested processes:5 5

Enter the size of memory block:
200 300 400 500 600

Enter the size of memory requested by process:
400 300 120 80 212

Enter 1:FirstFit, 2.BestFit, 3. WorstFit
Enter your choice:1

FirstFit
Process         MemoryBlock
1               3
2               2
3               1
4               1
5               4
Enter 1:FirstFit, 2.BestFit, 3. WorstFit
Enter your choice:2

BestFit
Process         MemoryBlock
1               3
2               2
3               1
4               1
5               4
Enter 1:FirstFit, 2.BestFit, 3. WorstFit
Enter your choice:3

WorstFit
Process         MemoryBlock
1               5
2               4
3               3
4               2
5               3
Enter 1:FirstFit, 2.BestFit, 3. WorstFit
Enter your choice:
```

6. Write a C program to implement the following page replacement algorithms:
a)FIFO b) LRU c) LFU

a)

```
#include<stdio.h>
int main()
{
	int i,j,n,a[50],frame[10],no,k,avail,count=0;
	printf("\n ENTER THE NUMBER OF PAGES:\n");
	scanf("%d",&n);
	printf("\n ENTER THE PAGE NUMBER :\n");
	for(i=1;i<=n;i++)
		scanf("%d",&a[i]);
	printf("\n ENTER THE NUMBER OF FRAMES :");
	scanf("%d",&no);
	for(i=0;i<no;i++)
		frame[i]= -1;
	j=0;
	printf("ref string\t\t page frames\n");
	for(i=1;i<=n;i++)
	{
		printf("%d\t\t",a[i]);
		avail=0;
		for(k=0;k<no;k++)
		if(frame[k]==a[i])
		avail=1;
		if (avail==0)
		{
			frame[j]=a[i];
			j=(j+1)%no;
			count++;
			for(k=0;k<no;k++)
			printf("%d\t",frame[k]);
		}
		printf("\n");
	}
	printf("Page Fault Is %d\n",count);
	return 0;
}
```

Output

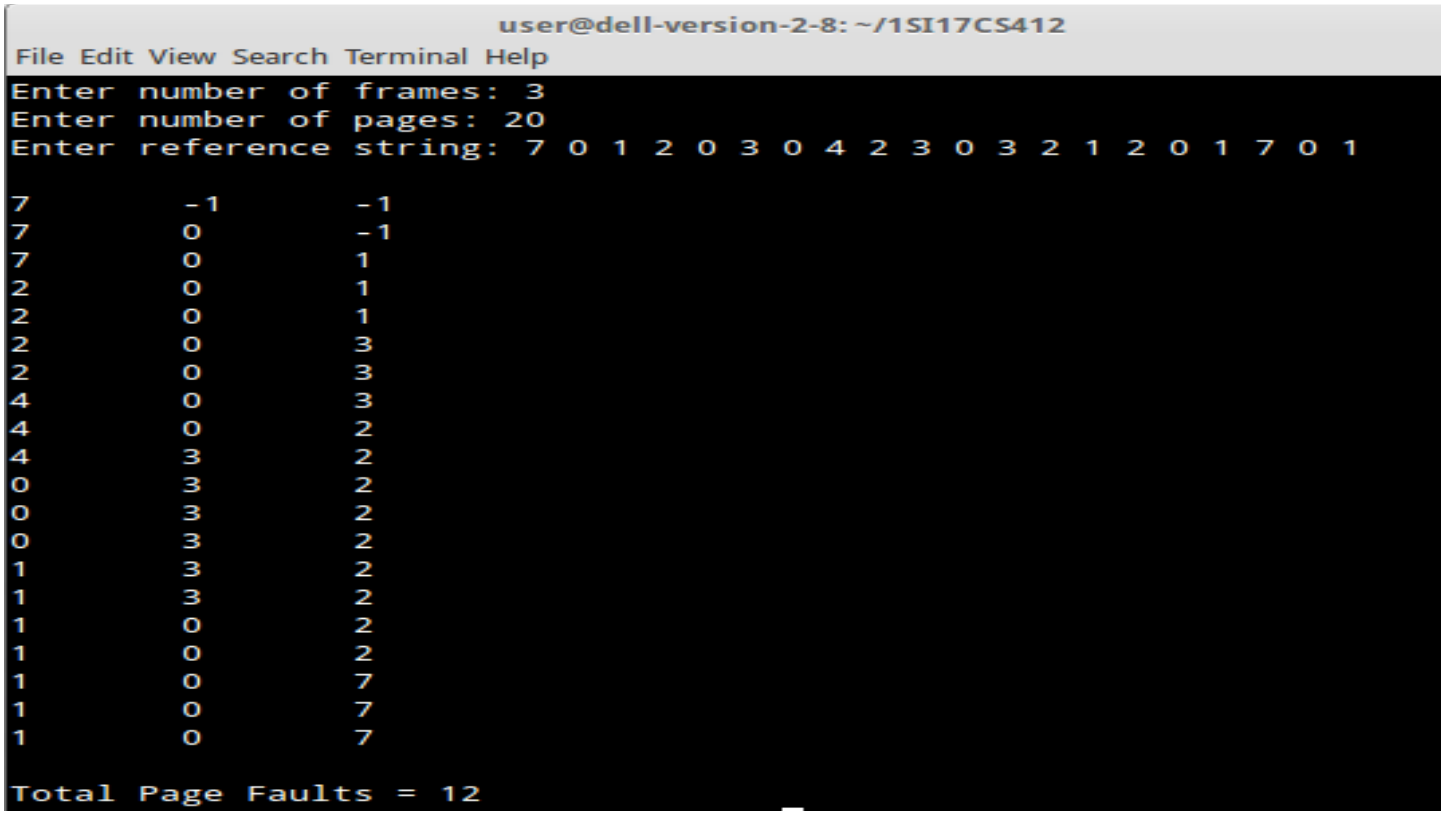
```
 ENTER THE NUMBER OF PAGES:
22

 ENTER THE PAGE NUMBER :
1 2 3 4 5 3 4 1 6 7 8 7 8 9 7 8 9 5 4 5 4 2

 ENTER THE NUMBER OF FRAMES :4
ref string               page frames
1               1       -1      -1      -1
2               1       2       -1      -1
3               1       2       3       -1
4               1       2       3       4
5               5       2       3       4
3
4
1               5       1       3       4
6               5       1       6       4
7               5       1       6       7
8               8       1       6       7
7
8
9               8       9       6       7
7
8
9
5               8       9       5       7
4               8       9       5       4
5
4
2               2       9       5       4
Page Fault Is 13
```

b) LRU

```
#include<stdio.h>
#include<stdlib.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 no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;
	printf("Enter number of frames: ");
	scanf("%d", &no_of_frames);
	printf("Enter number of pages: ");
	scanf("%d", &no_of_pages);
	printf("Enter reference string: ");
   for(i = 0; i < no_of_pages; ++i){
	scanf("%d", &pages[i]);
   }
	for(i = 0; i < no_of_frames; ++i){
	frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i){
	flag1 = flag2 = 0;
	for(j = 0; j < no_of_frames; ++j){
		if(frames[j] == pages[i]){
			counter++;
			time[j] = counter;
				flag1 = flag2 = 1;
				break;
			}
	}
	if(flag1 == 0){
			for(j = 0; j < no_of_frames; ++j){
			if(frames[j] == -1){
				counter++;
				faults++;
				frames[j] = pages[i];
				time[j] = counter;
				flag2 = 1;
				break;
			}
		}
	}
	if(flag2 == 0){
		pos = findLRU(time, no_of_frames);
		counter++;
		faults++;
		frames[pos] = pages[i];
		time[pos] = counter;
	}
	printf("\n");
	for(j = 0; j < no_of_frames; ++j){
		printf("%d\t", frames[j]);
	}
	}
```

```
        printf("\n\nTotal Page Faults = %d\n", faults);
    return 0;
}
```

OUTPUT :

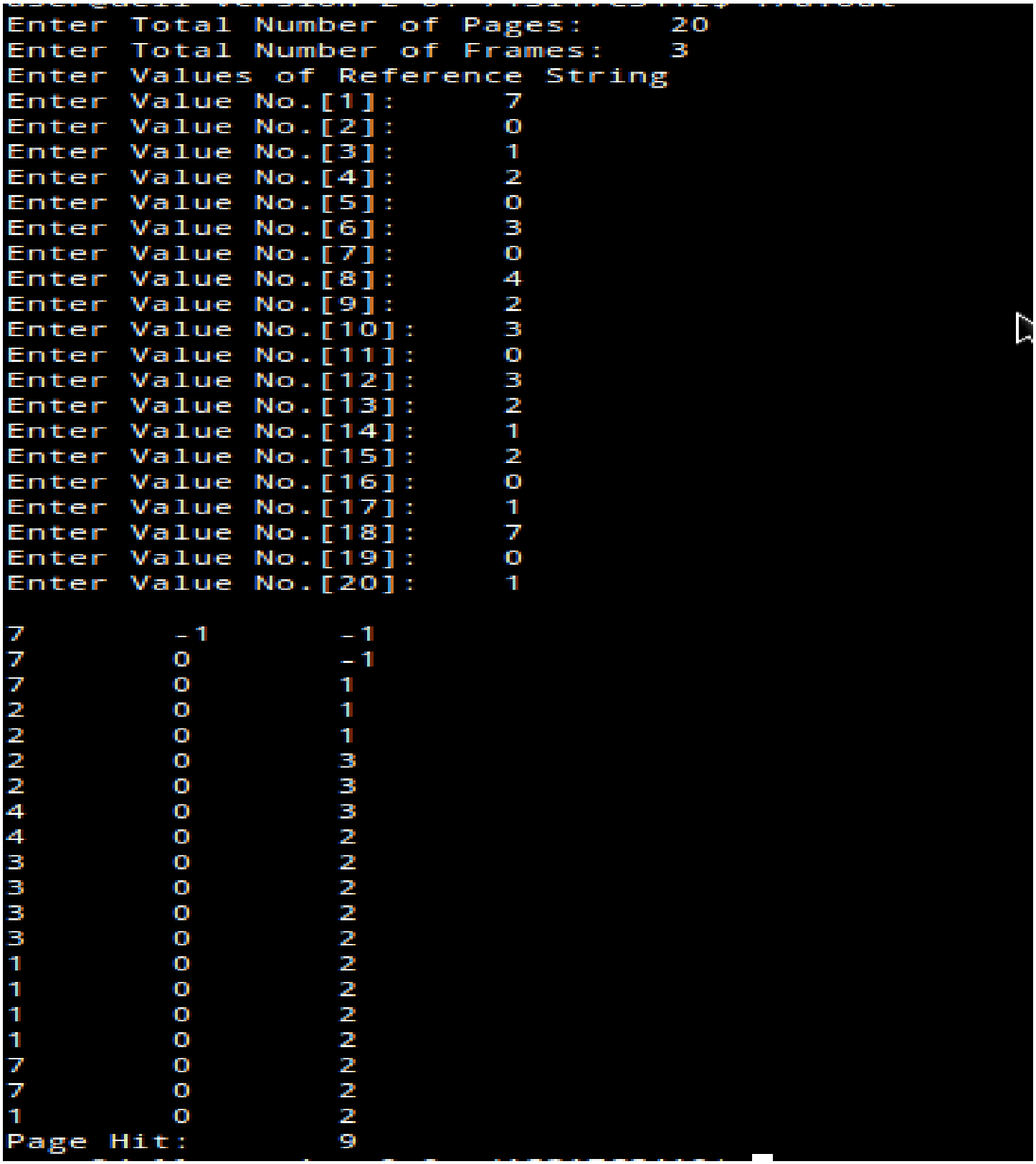


c) LFU

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int total_frames, total_pages, hit = 0;
    int pages[25], frame[10], arr[25], time[25];
    int m, n, page, flag, k, minimum_time, temp;
    printf("Enter Total Number of Pages:\t");
    scanf("%d", &total_pages);
    printf("Enter Total Number of Frames:\t");
    scanf("%d", &total_frames);
    for(m = 0; m < total_frames; m++)
    {
        frame[m] = -1;
    }
    for(m = 0; m < 25; m++)
    {
        arr[m] = 0;
    }
    printf("Enter Values of Reference String\n");
    for(m = 0; m < total_pages; m++)
    {
        printf("Enter Value No.[%d]:\t", m + 1);
        scanf("%d", &pages[m]);
    }
    printf("\n");
    for(m = 0; m < total_pages; m++)
    {
        arr[pages[m]]++;
        time[pages[m]] = m;
        flag = 1;
```

```
        k = frame[0];
        for(n = 0; n < total_frames; n++)
        {
            if(frame[n] == -1 || frame[n] == pages[m])
            {
                if(frame[n] != -1)
                {
                    hit++;
                }
                flag = 0;
                frame[n] = pages[m];
                break;
            }
            if(arr[k] > arr[frame[n]])
            {
                k = frame[n];
            }
        }
        if(flag)
        {
            minimum_time = 25;
            for(n = 0; n < total_frames; n++)
            {
                if(arr[frame[n]] == arr[k] && time[frame[n]] < minimum_time)
                {
                    temp = n;
                    minimum_time = time[frame[n]];
                }
            }
            arr[frame[temp]] = 0;
            frame[temp] = pages[m];
        }
        for(n = 0; n < total_frames; n++)
        {
            printf("%d\t", frame[n]);
        }
        printf("\n");
    }
    printf("Page Hit:\t%d\n", hit);
    return 0;
}
```

OUTPUT :



OR (using switch)

```
#include<stdio.h>
int n,nf,in[100],p[50],hit=0,i,j,k,pgfaultcnt=0;
void getData()
{
   printf("Enter length of page reference sequence\n");
   scanf("%d",&n);
   printf("Enter the page reference sequence\n");
   for(i=0; i<n; i++)
      scanf("%d",&in[i]);
   printf("Enter no of frames\n");
   scanf("%d",&nf);
```

```
}
void initialize()
{
   pgfaultcnt=0;
   for(i=0; i<nf; i++)
      p[i]=9999;
}
int isHit(int data)
{
   hit=0;
   for(j=0; j<nf; j++)
   {
      if(p[j]==data)
      {
         hit=1;
         break;
      }

   }
   return hit;
}
int getHitIndex(int data)
{
   int hitind;
   for(k=0; k<nf; k++)
   {
      if(p[k]==data)
      {
         hitind=k;
         break;
      }
   }
   return hitind;
}
void dispPages()
{
   for (k=0; k<nf; k++)
   {
      if(p[k]!=9999)
         printf(" %d",p[k]);
   }
}
void dispPgFaultCnt()    {          printf("\nTotal no of page faults:%d\n",pgfaultcnt); }
void fifo()
{
   initialize();
   for(i=0; i<n; i++)
   {
      printf("\nFor %d :",in[i]);

      if(isHit(in[i])==0)
      {

         for(k=0; k<nf-1; k++)
            p[k]=p[k+1];

         p[k]=in[i];
         pgfaultcnt++;
         dispPages();
      }
      else
         printf("No page fault");
   }
   dispPgFaultCnt();
```

```
}
void lru()
{
   initialize();
   int least[50];
   for(i=0; i<n; i++)
   {
      printf("\nFor %d :",in[i]);
      if(isHit(in[i])==0)
      {
         for(j=0; j<nf; j++)
         {
            int pg=p[j];
            int found=0;
            for(k=i-1; k>=0; k--)
            {
               if(pg==in[k])
               {
                  least[j]=k;
                  found=1;
                  break;
               }
               else
                  found=0;
            }
            if(!found)
               least[j]=-9999;
         }
         int min=9999;
         int repindex;
         for(j=0; j<nf; j++)
         {
            if(least[j]<min)
            {
               min=least[j];
               repindex=j;
            }
         }
         p[repindex]=in[i];
         pgfaultcnt++;
         dispPages();
      }
      else
         printf("No page fault");
   }
   dispPgFaultCnt();
}
void lfu()
{
   int usedcnt[100],least,repin,sofarcnt=0,bn;
   initialize();
   for(i=0; i<nf; i++)
      usedcnt[i]=0;
   for(i=0; i<n; i++)
   {
      printf("\n For %d :",in[i]);
      if(isHit(in[i]))
      {
         int hitind=getHitIndex(in[i]);
         usedcnt[hitind]++;
         printf("No page fault");
      }
      else
      {
```

```
            pgfaultcnt++;
            if(bn<nf)
            {
               p[bn]=in[i];
               usedcnt[bn]=usedcnt[bn]+1;
               bn++;
            }
            else
            {
               least=9999;
               for(k=0; k<nf; k++)
                  if(usedcnt[k]<least)
                  {
                     least=usedcnt[k];
                     repin=k;
                  }
               p[repin]=in[i];
               sofarcnt=0;
               for(k=0; k<=i; k++)
                  if(in[i]==in[k])
                     sofarcnt=sofarcnt+1;
               usedcnt[repin]=sofarcnt;
            }
            dispPages();
         }
      }
      dispPgFaultCnt();
}
int main()
{
   int choice;
   while(1)
   {
      printf("1.Enter data\t2.FIFO\t3.LRU\t4.LFU\nEnter your choice:");
      scanf("%d",&choice);
      switch(choice)
      {
          case 1:getData();
                                 break;
                        case 2:
                           fifo();
                           break;
                        case 3:
                           lru();
                           break;
                        case 4:
                           lfu();
                           break;
                        default:printf("invalid choice\n");
                           break;
      }
   }
}
```

OUTPUT :

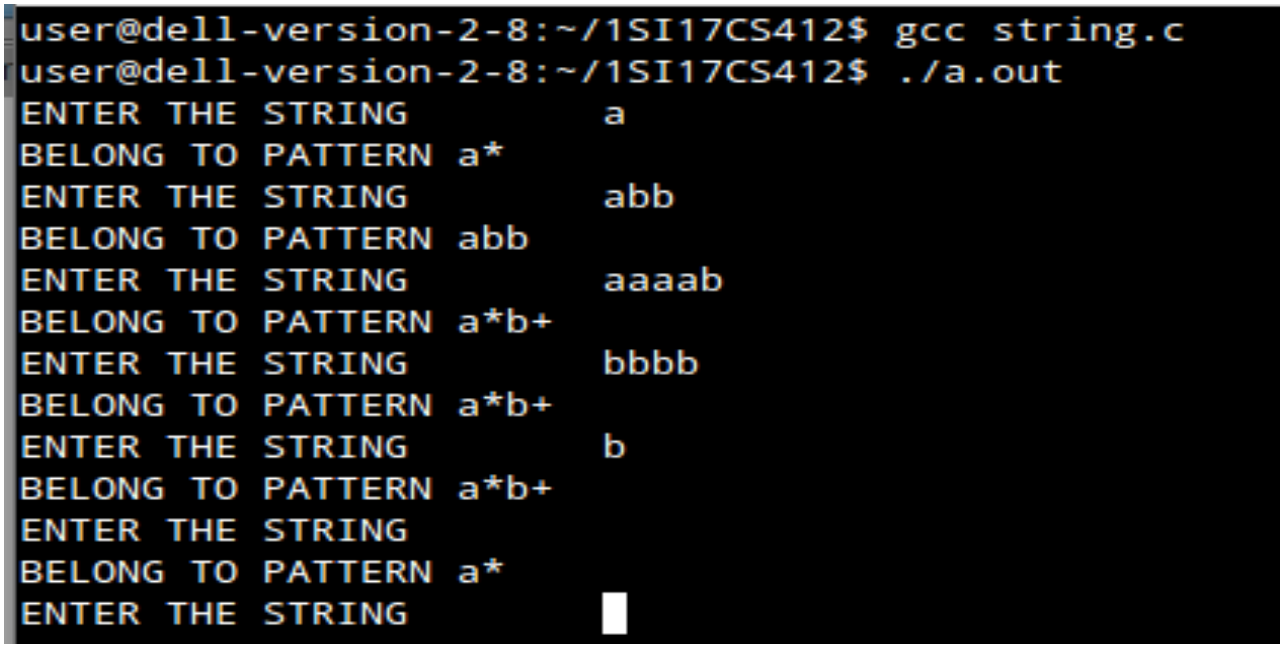
```
1.Enter data    2.FIFO  3.LRU   4.LFU
Enter your choice:1
Enter length of page reference sequence
20
Enter the page reference sequence
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter no of frames
3
1.Enter data    2.FIFO  3.LRU   4.LFU
Enter your choice:2

For 7 : 7
For 0 : 7 0
For 1 : 7 0 1
For 2 : 0 1 2
For 0 :No page fault
For 3 : 1 2 3
For 0 : 2 3 0
For 4 : 3 0 4
For 2 : 0 4 2
For 3 : 4 2 3
For 0 : 2 3 0
For 3 :No page fault
For 2 :No page fault
For 1 : 3 0 1
For 2 : 0 1 2
For 0 :No page fault
For 1 :No page fault
For 7 : 1 2 7
For 0 : 2 7 0
For 1 : 7 0 1
Total no of page faults:15
1.Enter data    2.FIFO  3.LRU   4.LFU
Enter your choice:
```

7. Write a C program to recognize strings under ‘a*’, ‘a*b+’, ‘abb’.

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
int main()
{
        char ch[100];
        int i,state;
        printf("ENTER THE STRING\t");
        gets(ch);
        i=0;
        state=0;
        while(ch[i]!='\0')
        {
                switch(state)
                {
                        case 0:if(ch[i]==' ')       state=0;
                                        else if(ch[i]=='a')
                                                state=1;
                                        else if(ch[i]=='b')
                                                state=2;
                                        else
                                                state=6;
                                        break;
                        case 1:if(ch[i]=='a')
                                                state=3;
                                        else if(ch[i]=='b')
                                                state=4;
                                        else
                                                state=6;
                                        break;
                        case 2:if(ch[i]=='a')
                                                state=6;
                                        else if(ch[i]=='b')
                                                state=2;
                                        else
```

```
                                        state=6;
                                break;
                        case 3:if(ch[i]=='a')
                                        state=3;
                                else if(ch[i]=='b')
                                        state=2;
                                else
                                        state=6;
                                break;
                        case 4:if(ch[i]=='a')
                                        state=6;
                                else if(ch[i]=='b')
                                        state=5;
                                else
                                        state=6;
                                break;
                        case 5:if(ch[i]=='a')
                                        state=6;
                                else if(ch[i]=='b')
                                        state=2;
                                else
                                        state=6;
                                break;
                        case 6:printf("NOT RECOGNISED\n");
                                return(main());
                }
                i++;
        }
        if(state==0 || state ==1 || state==3)
                printf("BELONG TO PATTERN a*\n");
        else if(state==2 || state==4)
                printf("BELONG TO PATTERN a*b+\n");
        else if(state==5)
                printf("BELONG TO PATTERN abb\n");
        else
                printf("NOT RECOGNISED\n");
        return (main());
}
```

Output :

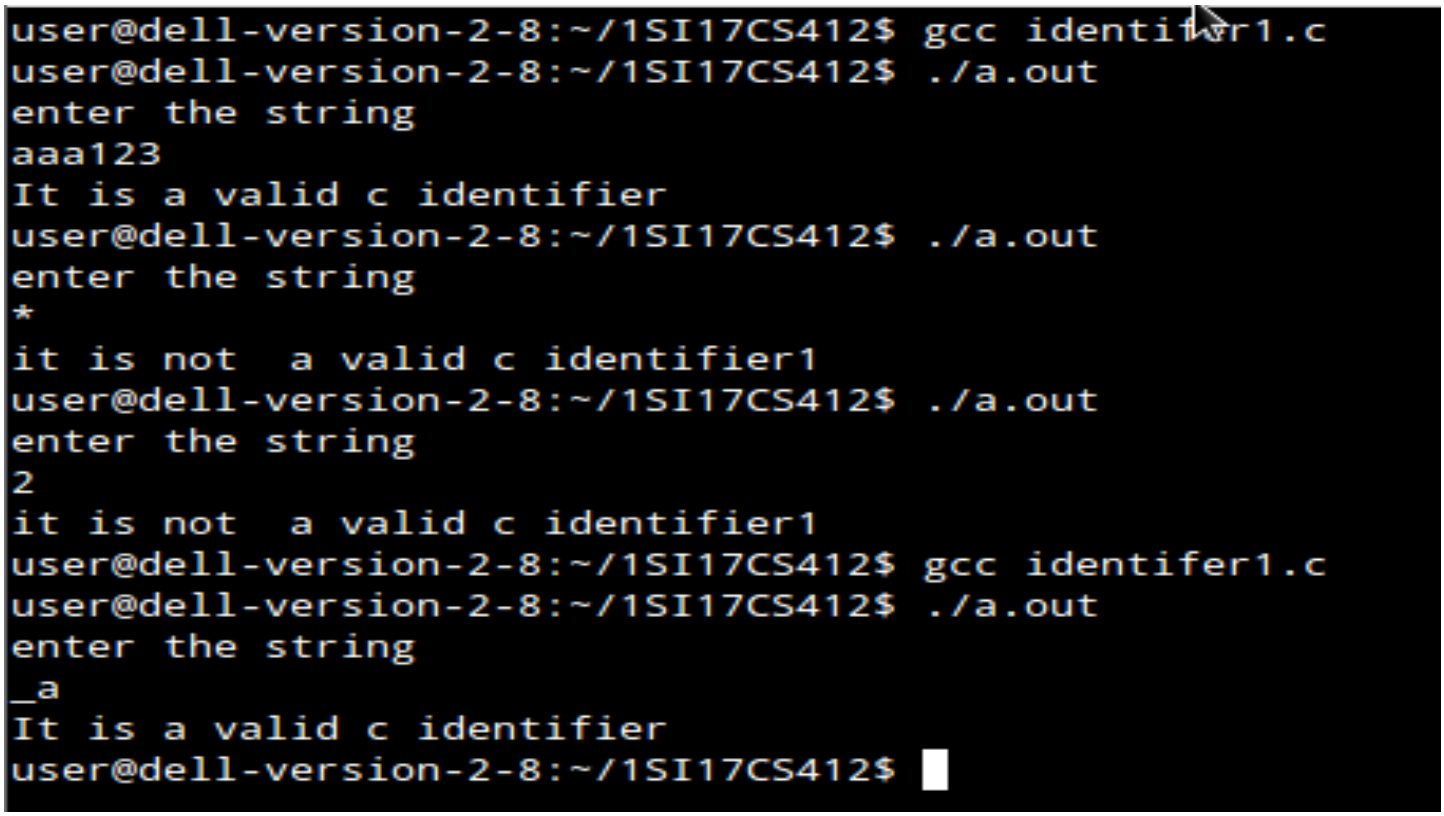


8. Write a C program to test whether a given identifier is valid or not.

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

```
#include<ctype.h>
int main()
{
	int i;
	char id[100];
	printf("enter the string\n");
	gets(id);
		if(id[0]!='_'&&!isalpha(id[0]))
	{
		printf("it is not  a valid c identifier1\n");
		return 0;
	}
	for(i=1;id[i]!='\0';i++)
	{
		if(isalpha(id[i])||isdigit(id[i])||id[i]=='_')
		{
			continue;
		}
		else
		{
		printf("It is  not a valid c identifier\n");
		return 0;
		}
	}
	printf("It is a valid c identifier\n");
	return 0;
}
```

Output :

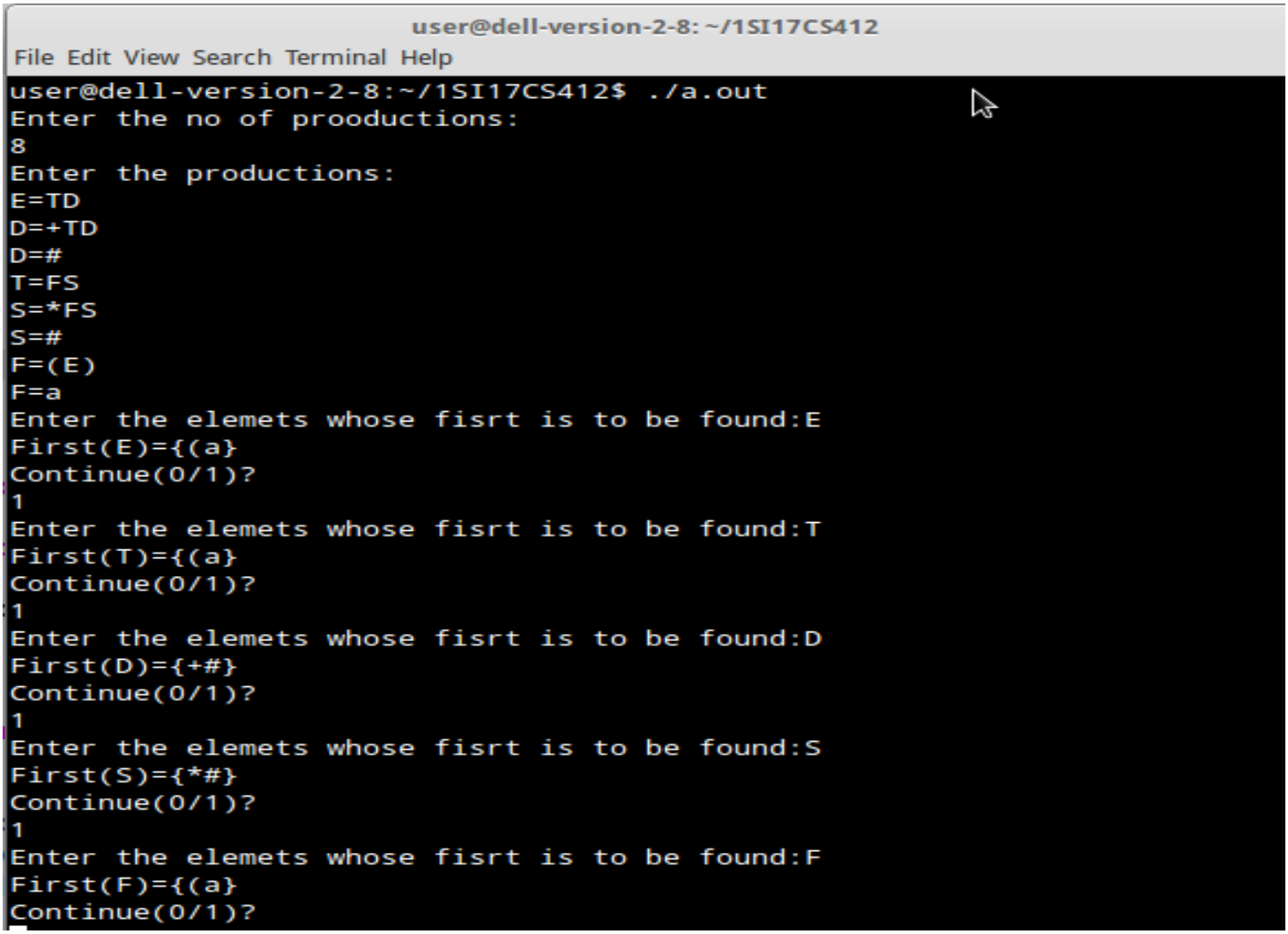


9. Write a C program to compute FIRST of all Non Terminals of a given grammar.

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
int n,m=0,p,i=0,j=0;
char a[10][10],f[10];
void first(char c);
```

```
int main()
{
	int i,z;
	char c,ch;
	printf("Enter the no of prooductions:\n");
	scanf("%d",&n);
	printf("Enter the productions:\n");
	for(i=0;i<n;i++)
	scanf("%s%c",a[i],&ch);
	do
	{
		m=0;
		printf("Enter the elemets whose fisrt is to be found:");
		scanf("%c",&c);
		first(c);
		printf("First(%c)={",c);
		for(i=0;i<m;i++)
		printf("%c",f[i]);
		printf("}\n");
		strcpy(f," ");
		printf("Continue(0/1)?\n");
		scanf("%d%c",&z,&ch);
	}
	while(z==1);
	return(0);
}
void first(char c)
{
	int k;
	if(!isupper(c))
	f[m++]=c;
	for(k=0;k<n;k++)
	{
		if(a[k][0]==c)
		{
			if(islower(a[k][2]))
				f[m++]=a[k][2];
			else first(a[k][2]);
		}
	}
}
```

OUTPUT :



10. Write a C program to construct predictive parsing table for the given grammar.

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
void addtonont(char);
void addtoter(char);
int nop,ppt[10][10];
char productions[10][10],ter[10],nont[10],first[10][10],follow[10][10];
int main()
{
        int i,j,k,m,pos=0;
        for(i=0;i<10;i++)
        for(j=0;j<10;j++)
        ppt[i][j]=-1;
        printf("Enter the number of productions:");
        scanf("%d",&nop);
        printf("\nEnter production like this eg:E->E+T Enter # for epsilon\n");
        for(i=0;i<nop;i++)
        {
                printf("Enter production number %d:",i+1);
                scanf("%s",productions[i]);
        }
        for(i=0;i<nop;i++)
        addtonont(productions[i][0]);
        for(i=0;i<nop;i++)
        for(j=3;productions[i][j]!='\0';j++)
        if(islower(productions[i][j])||(!isalpha(productions[i][j])))
        addtoter(productions[i][j]);
        for(j=0;ter[j]!='\0';j++);
```

```
ter[j]='$';
ter[++j]='\0';
printf("Enter first of all non terminals without any space b/w the symbols like abc#,#for epsilon\n" );
for(i=0;i<nop;i++)
{
	printf("Enter first of:");
	for(k=3;k<productions[i][k]!='\0';k++)
	printf("%c",productions[i][k]);
	printf("=");
	scanf("%s",first[i]);
	for(j=strlen(first[i]);j>=0;j--)
	first[i][j+1]=first[i][j];
	first[i][0]=productions[i][0];
}
printf("Enter follow of all non terminals without any space b/w symbols like abc#, # for epsilon\n");
for(i=0;nont[i]!='\0';i++)
{
	printf("Enter follow of %c=",nont[i]);
	scanf("%s",follow[i]);
	for(j=strlen(follow[i]);j>=0;j--)
	follow[i][j+1]=follow[i][j];
	follow[i][0]=nont[i];
}
for(i=0;i<nop;i++)
{
	for(m=0;follow[m][0]!=first[i][0];m++);
	for(j=1;first[i][j]!='\0';j++)
	if(first[i][j]!='#')
	{
		for(k=0;ter[k]!='\0';k++)
		if(ter[k]==first[i][j])
		break;
		ppt[m][k]=i;
	}
	else
	{
		for(m=0;follow[m][0]!=first[i][0];m++);
		for(j=1;follow[m][j]!='\0';j++)
		{
			for(k=0;ter[k]!='\0';k++)
			if(ter[k]==follow[m][j])
			break;
			ppt[m][k]=i;
		}
	}
	first[i][0]='0';
}
printf("Predictive parsing table\n");
printf(".....Terminals.....\n");
printf("Non Terminals |\t\t");
for(i=0;ter[i]!='\0';i++)
printf("%c\t",ter[i]);
printf("\n");
for(i=0;follow[i][0]!='\0';i++)
{
	m=0;
	printf("%c\t\t",nont[i]);
	for(j=0;ter[j]!='\0';j++)
	{
		pos=ppt[i][j];
		for(;m<=j;m++)
		printf("\t");
		if(pos!=-1)
		printf("%s",productions[pos]);
```

```
                }
                printf("\n");
        }
        return 0;
}
void addtonont(char c)
{
        int j;
        for(j=0;nont[j]!='\0';j++)
        if(nont[j]==c)
        return ;
        nont[j]=c;
        nont[j+1]='\0';
}
void addtoter(char c)
{
        int j;
        for(j=0;ter[j]!='\0';j++)
        if(ter[j]==c)
        return ;
        if(c!='#')
        {
                ter[j]=c;
                ter[j+1]='\0';
        }
}
```

OUTPUT :

```
Enter the  number of productions:8

Enter production like this eg:E->E+T Enter # for epsilon
Enter production number 1:E->TX
Enter production number 2:X->+TX
Enter production number 3:X->#
Enter production number 4:T->FY
Enter production number 5:Y->*FY
Enter production number 6:Y->#
Enter production number 7:F->i
Enter production number 8:F->(E)
Enter first of all non terminals without any space b/w the symbols like abc#,#fo
r epsilon
Enter first of:TX=+(
Enter first of:+TX=+
Enter first of:#=#
Enter first of:FY=i(
Enter first of:*FY=*
Enter first of:#=#
Enter first of:i=i
Enter first of:(E)=(
Enter follow of all non terminals without any space b/w symbols like abc#, # for epsilon
Enter follow of E=)$
Enter follow of X=)$
Enter follow of T=+)$
Enter follow of Y=+$)
Enter follow of F=+$)*
Predictive parsing table
..................Terminals.................
Non Terminals |         +       *       i       (       )       $
E                       E->TX                   E->TX
X                       X->+TX                          X->#    X->#
T                                       T->FY   T->FY
Y                       Y->#    Y->*FY                  Y->#    Y->#
F                                       F->i    F->(E)
user@dell-version-2-8:~/1SI17CS412$
```

11. Write a C program to implement recursive descent parsing for the given grammar.

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
void nonterminal(char );
int noofproduction,k,temp;
char productionset[20][20];
char str[20];
int reult;
int main()
{
        int i,ch;
        printf("\n Enter the no of productions:\n");
        scanf("%d",&noofproduction);
        printf("\n Enter the productions in form like E->E+T (Enter # for the epsilon)\n");
        for(i=0;i<noofproduction;i++)
        {
                printf("enter the production number %d:",i+1);
                scanf("%s",productionset[i]);
        }
        do
        {
                k=0;
                printf("\n Enter the string\n");
                scanf("%s",str);
                nonterminal(productionset[0][0]);
                if(k==strlen(str))
                        printf("\n input string is valid \n");
                else
                        printf("\n input string is invalid\n");
                printf("\n do you want to continue (0/1)\n");
                scanf("%d",&ch);
        }while(ch==1);
        return 0;

}
void nonterminal(char p)
{
   int i,j,found=0;
   for(i=0; i<noofproduction; i++)
   {
      temp=k;
      if(productionset[i][0]==p)
      {
         for(j=3; productionset[i][j]!='\0'; j++)
         {
            if(isupper(productionset[i][j]))
            {
               found=1;
               nonterminal(productionset[i][j]);
            }
            else if (productionset[i][j]==str[k])
            {
               k++;
               found=1;
            }
            else if(productionset[i][j]=='#')
            {
               found=1;
               return;
            }
            else
```

```
            {
               k=temp;
               break;
            }
      }
   }
   }
   if(i>=noofproduction && found==0 && k!=strlen(str))
   {
      printf("input invalid");
      exit(0);
   }
}
```

OUTPUT :

```
 Enter the no of productions:
3

 Enter the productions in form like E->E+T (Enter # for the epsilon)
enter the production number 1:S->cAd
enter the production number 2:A->ab
enter the production number 3:A->a

 Enter the string
cad

 input string is valid

 do you want to continue (0/1)
1

 Enter the string
caa

 input string is invalid

 do you want to continue (0/1)
0
user@dell-version-2-8:~/1SI17CS412$
```