## SPOS LAB 1 CODE

Design suitable data structures and implement pass-I of a two-pass assembler. Implementation should consist of a few instructions from each category and few assembler directives.

## ASSEMBLER.C

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
	char opcode[10], operand[10], label[10], code[10], mnemonic[3];
	int locctr, start, length;
	FILE *fp1, *fp3, *fp4, *fp2;

	fp1=fopen("input.txt", "r");
	fp2=fopen("optab.txt", "r");
	fp3=fopen("symtab.txt", "w");
	fp4=fopen("output.txt", "w");

	fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);

	if(strcmp(opcode, "START")==0)
	{
		start=atoi(operand);
		locctr=start;
		fprintf(fp4, "\t%s\t%s\t%s\n", label, opcode, operand);
		fscanf(fp1, "%s\t%s\t%s\n", label, opcode, operand);
	}
	else
		locctr=0;

	while(strcmp(opcode,"END")!=0)
	{
		fprintf(fp4, "%d\t", locctr);
		if (strcmp(label, "**")!=0)
			fprintf(fp3, "%s\t%d\n", label, locctr);
		fscanf(fp2, "%s\t%s", code, mnemonic);

		while(strcmp(code, "END")!=0)
		{
			if(strcmp(opcode, code)==0)
			{
				locctr+=3;
				break;
			}
			fscanf(fp2, "%s\t%s", code, mnemonic);
		}
```

```
            if(strcmp(opcode, "WORD")==0)
                   locctr+=3;

            else if(strcmp(opcode, "RESW")==0)
                   locctr+=(3*(atoi(operand)));
            else if (strcmp(opcode, "RESB")==0)
                   locctr+=atoi(operand);
            else if (strcmp(opcode, "BYTE")==0)
              ++locctr;

            fprintf(fp4, "%s\t%s\t%s\t\n", label, opcode, operand);
            fscanf(fp1, "%s\t%s\t%s", label, opcode, operand);
      }

      fprintf(fp4, "%d\t%s\t%s\t%s\n", locctr, label, opcode, operand);

      length=locctr-start;

      printf("The length of the code: %d\n", length);

      fclose(fp1);
      fclose(fp2);
      fclose(fp3);
      fclose(fp4);
            }
```

## INPUT.TXT

```
**      START   2000
**      LDA     FIVE
**      STA     ALPHA
**      LDCH    CHARZ
**      STCH    C1
ALPHA   RESW    2
FIVE    WORD    5
CHARZ   BYTE    C'Z'
C1      RESB    1
**      END     **
```

## OPTAB.TXT

```
START   *
LDA     03
STA     0F
LDCH    53
STCH    57
END     *
```

## SPOS LAB 2 CODE

Implement pass-II of a two-pass assembler. The output of assignment-1 (intermediate file and symbol table) should be input for this assignment.

## Corrected_Pass2

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int main()
{
char a[10],ad[10],label[10],opcode[10],operand[10],mnemonic[10],symbol[10];
int i,address,code,add,len,actual_len;
FILE *fp1,*fp2,*fp3,*fp4;
//clrscr();
system("cls");
fp1=fopen("assmlist.txt","w");
fp2=fopen("symtab.txt","r");
fp3=fopen("intermediate.txt","r");
fp4=fopen("optab.txt","r");
fscanf(fp3,"%s%s%s",label,opcode,operand);
if(strcmp(opcode,"START")==0)
{
fprintf(fp1,"\t%s\t%s\t%s\n",label,opcode,operand);
fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
}
while(strcmp(opcode,"END")!=0)
```

```
{
if(strcmp(opcode,"BYTE")==0)
{
fprintf(fp1,"%d\t%s\t%s\t%s\t",address,label,opcode,operand);
len=strlen(operand);
actual_len=len-3;
for(i=2;i<(actual_len+2);i++)
{
(operand[i],ad,16);
fprintf(fp1,"%s",ad);
}
fprintf(fp1,"\n");
}
else if(strcmp(opcode,"WORD")==0)
{
len=strlen(operand);
(atoi(operand),a,10);
fprintf(fp1,"%d\t%s\t%s\t%s\t00000%s\n",address,label,opcode,operand,a);
}
else if((strcmp(opcode,"RESB")==0)||(strcmp(opcode,"RESW")==0))
{
fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
}
else
{
rewind(fp4);
fscanf(fp4,"%s%d",mnemonic,&code);
while(strcmp(opcode,mnemonic)!=0)
```

```
fscanf(fp4,"%s%d",mnemonic,&code);
if(strcmp(operand,"**")==0)
{
fprintf(fp1,"%d\t%s\t%s\t%s\t%d0000\n",address,label,opcode,operand,code);
}
else
{
rewind(fp2);
fscanf(fp2,"%s%d",symbol,&add);
while(strcmp(operand,symbol)!=0)
{
fscanf(fp2,"%s%d",symbol,&add);
}
fprintf(fp1,"%d\t%s\t%s\t%s\t%d%d\n",address,label,opcode,operand,code,add);
}
}
fscanf(fp3,"%d%s%s%s",&address,label,opcode,operand);
}
fprintf(fp1,"%d\t%s\t%s\t%s\n",address,label,opcode,operand);
printf("Finished");
fclose(fp1);
fclose(fp2);
fclose(fp3);
fclose(fp4);
getch();
return 0;
}
```

**Intermediate.**txt

```
** START 2000
2000 ** LDA FIVE
2003 ** STA ALPHA
2006 ** LDCH CHARZ
2009 ** STCH C1
2012 ALPHA RESW 1
2015 FIVE WORD 5
2018 CHARZ BYTE C'EOF'
2019 C1 RESB 1
2020 ** END **
```

## SYMTAB.TXT

```
ALPHA 2012
FIVE 2015
CHARZ 2018
C1 2019
```

## OPTAB.txt

LDA 33
STA 44
LDCH 53
STCH 57
END *

## SPOS LAB 3

Design suitable data structures and implement macro definition and macro expansion processing for a sample macro with positional and keyword parameters.

## MACRO.TXT

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
char line[80],t1[10],t2[20],t3[10],FPN[20],APN[20],mname[10];
int count , v1,v2,v3,v4;
FILE *ifp;
int main()
{
int t21,t31,index=1;
ifp= fopen("int.txt","r");
while(!feof(ifp))
```

```
{
fgets(line,179,ifp);
count = sscanf(line,"%s%s%s",t1,t2,t3);
if(strcmp("MACRO",t1)==0)
{
strcpy(mname,t2);
printf("\n macro name table");
printf("\n \n");
}
if(strcmp(mname,t2)==0)
{
strcpy(FPN,t3);
printf("\n\n\n**FORMAL PARAMETER NAME TABLE**");
printf("\n :\n");
printf("\nINDEX\t\t:MACRO NAME");
printf("\n%d\t:%s",index,FPN);
}
if(strcmp(mname,t1)==0)
{
strcpy(APN,t2);
printf("\n\n\n**ACTUAL PARAMETER NAME TABLE**");
printf("\n :\n");
printf("\nINDEX\t\t:MACRO NAME");
printf("\n%d\t:%s",index,APN);
}
} }
```

**int.txt**

```
MACRO MIT &Y
ADD AREG BREG MEND


START 100
MOVER AREG CREG
MIT 10
SUB AREG CREG
END
```

## SPOS LAB 5

Process control system calls - Fork, execve and wait system calls along with the demonstration of zombie and orphan states.
a) Application should consist of Fork wait combination (parent with one application and child with another application) and students must demonstrate zombie and orphan states.

b) Application should consist of Fork execve combination (parent with one

application
and child with another application).

```
#include <stdio.h>
#include<sys/types.h>  //fork, sleep, getpid, getppid
#include<sys/wait.h>    //system call - wait
#include<stdlib.h>
#include<unistd.h>


int main()
{
        pid_t cpid;                  //Declaring a variable cpid with the
pid_t data type
        int  *status=NULL;    //Intilizating a pointer var status to NULL

        cpid = fork();               //The process fork_wait creates a new
process --clone

        if( cpid == 0 ) { //CHILD PROCESS as it is not creating any new
process
                printf("\n*************** This is child process
***************\n "); //write sys call

                printf("\n\t Process id is : %d", getpid());
                printf("\n\t Parent's process id is : %d", getppid());
                sleep(15);

                printf("\n*************Child process terminates
***************\n");
        }
        else { /*Parent process waiting for child process, to complete the
task*/

                printf("\n\t My process id is : %d", getpid());
                printf("\n\t My Parent process id is : %d", getppid());
                cpid = wait(status); //Forceful wait; that collects the
exit status of child process with cpid

                printf("\n\n\t Parent process collected the exit status of
child process with PID %d\n\n", cpid);
        }//end of if-else

        return 0;
}//end of main
```

## Execve code

```
//Execute ls command from the child process

#include <unistd.h>  //for execv
#include <sys/types.h>
```

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

int main()
{
	pid_t pid;
	int arr[10];
	char * argv_list[] = {"ls","-lrt","/home/hp/Botnet", NULL}; // ls -
lart /home/  NULL - no more arg

	pid=fork();

	if(pid==0)  //child process
	{
		execv("/bin/ls",argv_list); //arg1 - path of the
executable
		printf("\n This is child \n");
	}
	else //parent process
	{
		//sleep(30);
		printf("\n This is parent process\n");
	}
}
```

## Testfork.c

```
#include <stdio.h>
#include<sys/types.h> //return value of fork() -pid
#include<stdlib.h>
#include<unistd.h>


int main()
{
	pid_t cpid; //cpid is a var of data type pid_t i.e. int to store
(+ve, -ve or 0)

	cpid = fork();  //create the process and its id is stored in cpid

	if ( cpid < 0){
		printf("\n Cannot create the process");
		exit(0);
	}

	if( cpid == 0 ) { //CHILD PROCESS
		//sleep(50);  //to demonstrate orphan state
		printf("\n*************** This is child process
***************\n ");
		printf("\n\t Child process id is : %d", getpid());
		printf("\n\t My Parent process id is : %d\n", getppid());
		printf("\n*************Child process terminates
***************\n");
```

```
        }
        else { //Parent process
               sleep(30);  //to demonstrate zombie state
               printf("\n\n\t Parent process");
               printf("\n\t Parent process id is : %d\n", getpid());
               printf("\n\t My Parent process id is : %d\n", getppid());
        }//end of if-else



        return 0;
}//end of main
```

## SPOS LAB 6

Implement matrix multiplication using multithreading with pthread library.

Linux mai perform karna hai yeh walla experiment: pthread1.c (yehi name dena)

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

void *print_message_function( void *ptr );

main()
{
     pthread_t thread1, thread2;
     char *message1 = "Thread 1";
     char *message2 = "Thread 2";
     int  iret1, iret2;

    /* Create independent threads each of which will execute function */

     iret1 = pthread_create( &thread1, NULL, print_message_function, (void*)
message1);
     iret2 = pthread_create( &thread2, NULL, print_message_function, (void*)
message2);

     /* Wait till threads are complete before main continues. Unless we  */
     /* wait we run the risk of executing an exit which will terminate   */
     /* the process and all threads before the threads have completed.   */

     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     printf("Thread 1 returns: %d\n",iret1);
     printf("Thread 2 returns: %d\n",iret2);
     exit(0);
}

void *print_message_function( void *ptr )
{
```

```
    char *message;
    message = (char *) ptr;
    printf("%s \n", message);
}
```

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

#define M 3
#define K 2
#define N 3
#define NUM_THREADS 10

int A [M][K] = { {1,4}, {2,5}, {3,6} };
int B [K][N] = { {8,7,6}, {5,4,3} };
int C [M][N];

struct v {
  int i;
  int j;
};

void *runner(void *param);

int main(int argc, char *argv[]) {

  int i,j, count = 0;
  for(i = 0; i < M; i++) {
    for(j = 0; j < N; j++) {

      struct v *data = (struct v *) malloc(sizeof(struct v));
      data->i = i;
      data->j = j;

      pthread_t tid;
      pthread_attr_t attr;

      pthread_attr_init(&attr);

      pthread_create(&tid,&attr,runner,data);
```

```
        pthread_join(tid, NULL);
        count++;
      }
    }


    for(i = 0; i < M; i++) {
      for(j = 0; j < N; j++) {
        printf("%d ", C[i][j]);
      }
      printf("\n");
    }
}


void *runner(void *param) {
    struct v *data = param;
    int n, sum = 0;


    for(n = 0; n< K; n++){
      sum += A[data->i][n] * B[n][data->j];
    }

    C[data->i][data->j] = sum;


    pthread_exit(0);
}
```

**USE THIS MOST PROBABLY**

```
//Implement matrix multiplication using multithreading with pthread
library.

#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
#define MAX 3

int a[MAX][MAX];
int b[MAX][MAX];
int c[MAX][MAX];
```

```
//Generic function prototypes
void* mult(void*);

int main()
{
    pthread_t tid1,tid2,tid3; //Threads for row1, row2, and row3
    int row1,row2,row3;
    int i,j;

    printf("\n\nEnter First matrix : ");

    for(i=0;i<MAX;i++){
        for(j=0;j<MAX;j++){
            printf("\nEnter a[%d][%d] : ",i,j);
            scanf("%d",&a[i][j]);
        }
    }

    printf("\n\nEnter Second matrix");

    for(i=0;i<MAX;i++){
        for(j=0;j<MAX;j++){
            printf("\nEnter b[%d][%d] : ",i,j);
            scanf("%d",&b[i][j]);
        }
    }

    row1=0;
    //Create a thread as tid1; executing mult routine; accepting index of
row1
    pthread_create(&tid1, NULL, mult, &row1);  // pthread_create(a1, a2,
mult, 0)

    row2=1;
    //Create a thread as tid2; executing mult routine; accepting index of
row2
    pthread_create(&tid2, NULL, mult, &row2);

    row3=2;
    //Create a thread as tid3; executing mult routine; accepting index of
row3
    pthread_create(&tid3, NULL, mult, &row3);


    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    pthread_join(tid3,NULL);



    printf("\n\nResult is : \n");

    for(i=0;i<MAX;i++){
```

```
        for(j=0;j<MAX;j++){
           printf("%d ",c[i][j]);
        }
        printf("\n");
     }

   exit(0);

}//End of main


void* mult(void * arg)  // Address of 0th row
{
        int i,j,k;
        i = *(int * )arg;  //value of i = 0, 1, or 2

        for(j=0;j<MAX;j++){
                c[i][j] = 0;
                for(k=0;k<MAX;k++){
                        c[i][j] += a[i][k] * b[k][j];
                }
        }
        printf("\nThread id is : %ld",pthread_self());
}//END
```

## SPOS LAB 7

Simulation of following scheduling algorithms

FCFS (Non-preemptive by default)
SRTN (Preemptive version of SJF)
Priority (Non-preemptive)
Round Robin (Preemptive by default)

## FCFS

```
#include<stdio.h>

void findWaitingTime(int processes[], int n,
                                                int bt[], int wt[])
{

        wt[0] = 0;


        for (int i = 1; i < n ; i++ )
                wt[i] = bt[i-1] + wt[i-1] ;
}
```

```
void findTurnAroundTime( int processes[], int n,
                            int bt[], int wt[], int tat[])
{

        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}


void findavgTime( int processes[], int n, int bt[])
{
        int wt[n], tat[n], total_wt = 0, total_tat = 0;


        findWaitingTime(processes, n, bt, wt);


        findTurnAroundTime(processes, n, bt, wt, tat);


        printf("Processes Burst time Waiting time Turn around time\n");


        for (int i=0; i<n; i++)
        {
                total_wt = total_wt + wt[i];
                total_tat = total_tat + tat[i];
                printf(" %d ",(i+1));
                printf("        %d ", bt[i] );
                printf("        %d",wt[i] );
                printf("        %d\n",tat[i] );
        }
        int s=(float)total_wt / (float)n;
        int t=(float)total_tat / (float)n;
        printf("Average waiting time = %d",s);
        printf("\n");
        printf("Average turn around time = %d ",t);
}


int main()
{

        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];


        int burst_time[] = {10, 5, 8};

        findavgTime(processes, n, burst_time);
        return 0;
}
```

## SGF

```
#include <stdio.h>
int main()
{
    int A[100][4]; // Matrix for storing Process Id, Burst
                   // Time, Average Waiting Time & Average
                   // Turn Around Time.
    int i, j, n, total = 0, index, temp;
    float avg_wt, avg_tat;
    printf("Enter number of process: ");
    scanf("%d", &n);
    printf("Enter Burst Time:\n");
    // User Input Burst Time and alloting Process Id.
    for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &A[i][1]);
        A[i][0] = i + 1;
    }
    // Sorting process according to their Burst Time.
    for (i = 0; i < n; i++) {
        index = i;
        for (j = i + 1; j < n; j++)
            if (A[j][1] < A[index][1])
                index = j;
        temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;

        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
    }
    A[0][2] = 0;
    // Calculation of Waiting Times
    for (i = 1; i < n; i++) {
        A[i][2] = 0;
        for (j = 0; j < i; j++)
            A[i][2] += A[j][1];
        total += A[i][2];
    }
    avg_wt = (float)total / n;
    total = 0;
    printf("P     BT     WT     TAT\n");
    // Calculation of Turn Around Time and printing the
    // data.
    for (i = 0; i < n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        total += A[i][3];
        printf("P%d     %d     %d      %d\n", A[i][0],
               A[i][1], A[i][2], A[i][3]);
    }
    avg_tat = (float)total / n;
    printf("Average Waiting Time= %f", avg_wt);
    printf("\nAverage Turnaround Time= %f", avg_tat);
}
```

## PRIORITY

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
    int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
    int totwt=0,totta=0;
    float awt,ata;
    char pn[10][10],t[10];
    //clrscr();
    printf("Enter the number of process:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("Enter process name,arrivaltime,execution time &
priority:");
        //flushall();
        scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
    }
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
        {
            if(p[i]<p[j])
            {
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
                temp=at[i];
                at[i]=at[j];
                at[j]=temp;
                temp=et[i];
                et[i]=et[j];
                et[j]=temp;
                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }
        }
    for(i=0; i<n; i++)

    {

        if(i==0)
        {
            st[i]=at[i];
            wt[i]=st[i]-at[i];
            ft[i]=st[i]+et[i];
            ta[i]=ft[i]-at[i];
        }
        else
```

```
        {
            st[i]=ft[i-1];
            wt[i]=st[i]-at[i];
            ft[i]=st[i]+et[i];
            ta[i]=ft[i]-at[i];
        }
        totwt+=wt[i];
        totta+=ta[i];
    }
    awt=(float)totwt/n;
    ata=(float)totta/n;

printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime
");
    for(i=0; i<n; i++)

printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i
],ta[i]);
    printf("\nAverage waiting time is:%f",awt);
    printf("\nAverage turnaroundtime is:%f",ata);
    getch();
}
```

**RR**

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

void main()
{

   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;


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");
scanf("%d", &at[i]);
printf(" \nBurst time is: \t");
```

```
scanf("%d", &bt[i]);
temp[i] = bt[i];
}

printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);

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)
{
   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--;
      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;
    }
}

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();
}
```

**SPOS 8**

**Simulation of Banker's Algorithm**

**USER INPUT CODE**

```
#include<stdio.h>
void main()
{
    int n, m, i, j, k, alloc[20][20], max[20][20], available[20];
    int f[20],ans[20], ind=0, need[20][20];


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


    printf("Enter the number of Resources: ");
    scanf("%d",&m);


    printf("Enter the Values of Allocation Matrix: \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<m;j++)
        {
            printf("Enter value at position (%d%d) :",i+1,j+1);
            scanf("%d",&alloc[i][j]);
        }
    }
```

```
    printf("Enter the Values of Max Matrix: \n");
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
    {
        printf("Enter value at position (%d%d) :",i+1,j+1);
        scanf("%d",&max[i][j]);
    }
}


printf("Enter the values in available: \n");
for(j=0;j<m;j++)
{
    printf("Enter value at position (%d) :",j+1);
    scanf("%d",&available[j]);
}


for(k=0;k<n;k++)
{
    f[k]=0;
}


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<n;k++)
{
```

```
        for(i=0;i<n;i++)
        {
            if(f[i]==0)
            {
                int flag = 0;
                for(j=0;j<m;j++)
                {
                    if(need[i][j] > available[j])
                    {
                        flag=1;

                        break;
                    }
                }

                if(flag==0)
                {
                    ans[ind++] = i;
                    for(y=0;y<m;y++)
                    {
                        available[y] = available[y] + alloc[i][y];
                    }
                    f[i]=1;
                }
            }
        }
    }


    printf("The SAFE SEQUENCE is: \n");
    for(i=0;i<n-1;i++)
    {
        printf(" P%d ->", ans[i]);
    }
    printf(" P%d", ans[n-1]);

}
```

## ALREADY PUTTED VALUE CODE

```
#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);

    // This code is contributed by Deep Baldha (CandyZack)
}
```

## SPOS 10

Implement the following page replacement algorithms
- FIFO
- LRU
- Optimal

## FIFO

```
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
  printf("Value No. [%d]:\t", m + 1);
```

```
   scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
   scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
  temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
  s = 0;
  for(n = 0; n < frames; n++)
   {
      if(referenceString[m] == temp[n])
         {
            s++;
            pageFaults--;
         }
   }
   pageFaults++;
   if((pageFaults <= frames) && (s == 0))
      {
        temp[m] = referenceString[m];
      }
   else if(s == 0)
      {
        temp[(pageFaults - 1) % frames] = referenceString[m];
      }
      printf("\n");
      for(n = 0; n < frames; n++)
       {
         printf("%d\t", temp[n]);
       }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```

**ALGO**

1. Start traversing the pages.
2. Now declare the size w.r.t length of the Page.
3. Check need of the replacement from the page to memory.
4. Similarly, Check the need of the replacement from the old page to new page in memory.
5. Now form the queue to hold all pages.
6. Insert Require page memory into the queue.
7. Check bad replacemets and page faults.
8. Get no of processes to be inserted.
9. Show the values.
10. Stop

**LRU**

```
#include <stdio.h>

 //user-defined function

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;
}

//main function
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("\nTotal Page Faults = %d", faults);

  return 0;
}
```

**OPTIMAL**

```
#include<stdio.h>
                    void optimal(int string[20],int n,int size)
```

```
{
    //Creating array for block storage
    int frames[n];
    //Initializing each block with -1
    for (int i=0;i<n;i++)
        frames[i]=-1;


    //Index to insert element
    int index=-1;


    //Counters
    int page_miss=0;
    int page_hits=0;


    //Pointer to indicate initially frames filled
or not
    int full=0;


    //Traversing each symbol in fifo
    for (int i=0;i<size;i++)
    {
        int symbol=string[i];
        int flag=0;


        for(int j=0;j<n;j++)
        {
            if (symbol==frames[j])
            {
                flag=1;
                break;
            }
        }


        if (flag==1)
        {
            printf("\nSymbol: %d  Frame:
",symbol);
```

```
                for (int j=0;j<n;j++)
                    printf("%d ",frames[j]);
                page_hits+=1;
            }
            else
            {
                //Frames are still empty
                if (full==0)
                {
                    index=(index+1)%n;
                    frames[index]=symbol;
                    page_miss+=1;
                    printf("\nSymbol: %d  Frame:
",symbol);
                    for (int j=0;j<n;j++)
                        printf("%d ",frames[j]);


                    //Frames filled or not
                    if (i==n-1)
                        full=1;
                }


                //Frames are full, now we can apply
optimal page replacement
                else
                {
                    //First find the index to replace
with
                    int pos=-1;
                    int index=-1;


                    //Traversing each symbol and
checking their optimal possibility
                    for(int j=0;j<n;j++)
                    {
                        //Whether symbol in frame
found or not in future cached frame
                        int found=0;
                        for (int k=i+1;k<size;k++)
                        {
```

```
                        //If symbol exists in
cached string
                        if (frames[j]==string[k])
                        {
                            found=1;
                            if (pos<k)
                            {
                                pos=k;
                                index=j;
                            }
                            break;
                        }
                    }
                    //Symbol does not exist in
cached frame
                    if (found==0)
                    {
                        pos=size;
                        index=j;
                    }
                }


                //Now assign symbol in lru
position
                frames[index]=symbol;


                printf("\nSymbol: %d  Frame:
",symbol);
                for (int j=0;j<n;j++)
                    printf("%d ",frames[j]);
            }
        }
    }
    printf("\nPage hits: %d",page_hits);
    printf("\nPage misses: %d",page_miss);
}


//Main function
int main(void)
{
```

```
    int string[]={2, 3, 4, 2, 1, 3, 7, 5, 4, 3};
    int no_frames=3;
    int size=sizeof(string)/sizeof(int);
    optimal(string,no_frames,size);
    return 0;
}
```

## SPOS 4

```
#!/bin/bash
# creating a menu with the following options
echo "MENU DRIVEN PROGRAM";
echo "1. Factorial "
echo "2. Greatest "
echo "3. Prime Number "
echo "4. Number Palindrome "
echo "5. String palindrome "
echo "6. Exit from menu "
echo -n "Enter your menu choice [1-6]: "
# Running a forever loop using while statement
# This loop will run untill select the exit option.
# User will be asked to select option again and again
while :
do
# reading choice
read choice
# case statement is used to compare one value with the multiple cases.
case $choice in
# Pattern 1
1) echo -n "Enter a number: "
read number
factorial=1

for(( i=1; i<=number; i++ ))
do
```

```
factorial=$[ $factorial * $i ]
done
echo "The factorial of $number is $factorial"
# Pattern 2
echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
echo $num1
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
echo $num2
else
echo $num3
fi
# Pattern 3
 echo -e "Enter Number : \c"
read n
for((i=2; i<=$n/2; i++))
do
ans=$(( n%i ))
if [ $ans -eq 0 ]
then
echo "$n is not a prime number."
# exit 0
fi
done
echo "$n is a prime number."
# Pattern 4
 echo -n "Enter number : "
read n
# store single digit
sd=0
# store number in reverse order
rev=""
# store original number
on=$n
while [ $n -gt 0 ]
do
sd=$(( $n % 10 )) # get Remainder
n=$(( $n / 10 )) # get next digit
```

```
# store previous number and current digit in reverse
rev=$( echo ${rev}${sd} )
done
if [ $on -eq $rev ];
then
echo "Number is palindrome"
else
echo "Number is NOT palindrome"
fi
# Pattern 5
 #clear
echo "Enter a string to be entered:"
read str
echo
len=`echo $str | wc -c`
len=`expr $len - 1`
i=1
j=`expr $len / 2`
while test $i -le $j
do
k=`echo $str | cut -c $i`
l=`echo $str | cut -c $len`
if test $k != $l
then
echo "String is not palindrome"
exit
fi
i=`expr $i + 1`
len=`expr $len - 1`
done
echo "String is palindrome"
# Pattern 6
echo "Quiting ..."
exit

# Default Pattern


echo "invalid option"

esac
echo -n "Enter your menu choice [1-6]: "
done
```

file ka name assin.sh rkh lena

UBUNTU OPEN KRKE ye desktop m save kr lena then phir right click krke run in terminal krna then ye command chmod u+x filename.sh and ./filename.sh krke run kr dena

**SPOS 9**

```
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>

sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)
{
sem_wait(&wrt);
cnt = cnt*2;
printf("Writer %d modified cnt to %d\n",(*((int *)wno)),cnt);
sem_post(&wrt);
}
void *reader(void *rno)
{

pthread_mutex_lock(&mutex);
numreader++;
if(numreader == 1) {
sem_wait(&wrt);
}
pthread_mutex_unlock(&mutex);

printf("Reader %d: read cnt as %d\n",*((int *)rno),cnt);

pthread_mutex_lock(&mutex);
numreader--;
if(numreader == 0) {
sem_post(&wrt);
```

```
}
pthread_mutex_unlock(&mutex);
}
int main()
{
pthread_t read[10],write[5];
pthread_mutex_init(&mutex, NULL);
sem_init(&wrt,0,1);
int a[10] = {1,2,3,4,5,6,7,8,9,10};
for(int i = 0; i < 10; i++) {
pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
}
for(int i = 0; i < 5; i++) {
pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
}
for(int i = 0; i < 10; i++) {
pthread_join(read[i], NULL);
}
for(int i = 0; i < 5; i++) {
pthread_join(write[i], NULL);
}

pthread_mutex_destroy(&mutex);
sem_destroy(&wrt);
return;
}
```