/*OpenMP Programs*/

```
#include <stdio.h>
#include <omp.h>
main()
{
        int i, m, k;
        omp_set_dynamic(0);
        m= omp_get_num_procs();
        k=omp_get_max_threads();
        printf("\nno. of processors available for openmp=%d", m);
        printf("\nmax no. of threads=%d", k);
        omp_set_num_threads(6);
        #pragma omp parallel
        printf("\n Hello %d of %d", omp_get_thread_num(), omp_get_num_threads());
        fgetc(stdin);
}
```

```
#include <stdio.h>
#include <omp.h>
main()
{
        int A[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, i, m, k;
        omp_set_dynamic(0);
        m= omp_get_num_procs();
        omp_set_num_threads(m);
        #pragma omp parallel for shared(A) private(i)
        for(i=0;i<10;i++)
                printf("\nA[%d]= %d from thread %d of %d", i, A[i], omp_get_thread_num(),
omp_get_num_threads());
        fgetc(stdin);
}
```

```
#include <stdio.h>
#include <omp.h>
main()
{
	int A[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, i, sum;
	#pragma omp parallel for reduction(+: sum)
	for(i=0;i<10;i++)
		sum+=A[i];
	printf("\nSum = %d\n", sum);
	fgetc(stdin);
}
```

```
#include <stdio.h>
#include <omp.h>
main()
{
	int A[3][3]={1, 1, 1,
				1, 1, 1,
				1, 1, 1};
	int B[3][3]={2, 2, 2,
				2, 2, 2,
				2, 2, 2};
	int C[3][3], i, j, k;

	#pragma omp parallel for
		for(i=0;i<3;i++)
			for(j=0;j<3;j++)
				C[i][j]=0;

	for(i=0;i<3;i++)
		for(j=0;j<3;j++)
			#pragma omp parallel for
```

```
                    for(k=0;k<3;k++)
                         C[i][j]=C[i][j]+A[i][k]*B[k][j];

          for(i=0;i<3;i++)
          {
               printf("\n");
               for(j=0;j<3;j++)
                    printf(" %d", C[i][j]);
          }
          fgetc(stdin);
}
```

```
#include <stdio.h>
#include <omp.h>
main()
{
     int sum, i;
     #pragma omp parallel for reduction (+:sum)
     for(i=1;i<=5;i++)
          sum+=i*(i+1);
     printf("\n Sum = %d", sum);
     fgetc(stdin);
}
```

```
#include <stdio.h>
#include <omp.h>
main()
{
     #pragma omp parallel sections
     {
          #pragma omp section
```

```
                printf("\nFirst %d", omp_get_thread_num());
            #pragma omp section
                printf("\nSecond %d", omp_get_thread_num());
            #pragma omp section
                printf("\nThird %d", omp_get_thread_num());
        }
        fgetc(stdin);
}
```

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

int x=5;
first()
{
        x++;
        printf("\nx=%d from first", x);
}
second()
{
        x--;
        printf("\nx=%d from second", x);
}
main()
{
        #pragma omp parallel sections
        {
            #pragma omp section
            first();
            #pragma omp section
            second();
        }
        printf("\nx=%d at end", x);
        fgetc(stdin);
}
```

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

int x=5;
first()
{
        #pragma omp critical
        {
                x++;
                printf("\nx=%d from first", x);
        }
}

second()
{
        #pragma omp critical
        {
                x--;
                printf("\nx=%d from second", x);
        }
}

main()
{
        #pragma omp parallel sections
        {
                #pragma omp section
                        first();
                #pragma omp section
                        second();
        }
        printf("\nx=%d at end", x);
        fgetc(stdin);
}
```

```
#include <stdio.h>
#include <omp.h>
main()
{       int a=0, b=0;
        #pragma omp parallel num_threads(4)
        {
                #pragma omp single
                a++;
                #pragma omp critical
                b++;
        }
        printf("single: %d -- critical: %d\n", a, b);
        fgetc(stdin);
}
```

//Producer-Consumer Problem

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

main()
{
        int Q[100], front=0, rear=-1, count=0;
        int id, d, ins=0;
        omp_set_dynamic(0);
        #pragma omp parallel num_threads(2)
        {
                id=omp_get_thread_num();
                if(id==0)/*Producer*/
                while(1)
                {
                        #pragma omp critical
                        {
                                if(count<100)
                                {
                                        rear=(rear+1)%100;
                                        ins++;
                                        Q[rear]=ins;
                                        printf("\nProducer %d", ins);
                                        count++;
                                }
                                else
                                        printf("\nProducer no space");
                                fgetc(stdin);
                        }
                }
                else
                {
                        while(1)/*Consumer*/
                        {
                                #pragma omp critical
                                {
                                        if(count!=0)
                                        {
                                                d=Q[front];
                                                front=(front+1)%100;
                                                printf("\nConsumer %d", d);
                                                count--;
                                        }
                                        else printf
                                                ("\nConsumer no items");
                                fgetc(stdin);
                                }
                        }
                }
        }
}
```

//Producer-Consumer Problem with sched_yield()

```
#include <stdio.h>

#include <omp.h>

#include <sched.h>


main()

{

        int Q[100], front=0, rear=-1, count=0;

        int id, d, ins=1;

        omp_set_dynamic(0);

        #pragma omp parallel num_threads(2)

        {

                id=omp_get_thread_num();

                if(id==0)/*Producer*/

                while(1)

                {

                        #pragma omp critical

                        {

                                if(count<10)

                                {

                                        rear=(rear+1)%10;

                                        Q[rear]=ins;

                                        printf("\nProducer %d", ins++);

                                        count++;

                                }

                                else

                                        sched_yield();//nice

                                fgetc(stdin);

                        }

                }
                else
                {
                        while(1)/*Consumer*/
                        {
                                #pragma omp critical
                                {
                                        if(count!=0)
                                        {
                                                d=Q[front];
                                                front=(front+1)%10;
                                                printf("\nConsumer %d", d);
                                                count--;
                                        }
                                        else
                                                sched_yield();
                                        fgetc(stdin);
                                }
                        }
                }
        }
}
```