Q1- Write a multithreaded program that generates the Fibonacci series. This program should work as follows: The user will enter on the command line the number of Fibonacci numbers that the program is to generate. The program will then create a separate thread that will generate the Fibonacci numbers. When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, this will require having the parent thread wait for the child thread to finish.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
pthread_mutex_t mutex;
int fib[10];
int in = 0;
void *genFibo(void *param);
int main(int argc, char *argv[])
{
	pthread_attr_t attr;
	if (argc != 2)
	{
		fprintf(stderr,"usage: fibthread <integer value>\n");
		return -1;
	}
	int count = atoi(argv[1]);
	if (count < 1)
	{
		fprintf(stderr,"%d must be>= 1\n", count);
		return -1;
	}
	pthread_attr_init(&attr);
	pthread_mutex_init(&mutex, NULL);

	for(int i = 1;i <= count;i++)
	{
		pthread_t thread;
		pthread_create(&thread, &attr, genFibo, (void*)i);
		pthread_join(thread, NULL);	}
```

```
	for (int i = 0; i < in;i++)
	{
		printf("%d ", fib[i]);
	}
	printf("\n");
	pthread_mutex_destroy(&mutex);
}

void *genFibo(void *param)
{
	pthread_mutex_lock(&mutex);
	fib[in++] = fibonacci((int)param);
	pthread_mutex_unlock(&mutex);
	pthread_exit(0);
}

int fibonacci (int x)
{
	if (x <= 1)
	{
		return 1;
	}
	return fibonacci(x-1) + fibonacci(x-2);
}
```

Q2- A C program to show multiple threads with local, global and static variables. Create three threads and print the value of the static, global and local variable.

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

int g = 0; //Global variable

void *myThreadFun(void *vargp)
{
   int *myid = (int *)vargp;

   static int s = 0; // static variable

   ++s; ++g;

   printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);
}

int main()
{
   int i;
   pthread_t tid;

   // Let us create three threads
   for (i = 0; i < 3; i++)
      pthread_create(&tid, NULL, myThreadFun, (void *)&tid);

   pthread_exit(NULL);
   return 0;
}
```