Tutorial 9 – Threads

1. Copy the following program into a cpp file and compile it (using "g++ -pthread –o outputName source.cpp"). Modify it so that SleepTime is called on a separate thread. **Ensure you add your join in the correct place.**

```
#include<cstdlib>
#include<iostream>
#include<unistd.h>
#include<pthread.h>


using namespace std;

void *SleepTime(void *timeToSleep)
{
	int duration = *((int*)timeToSleep);
	sleep(duration);
	cout << "Slept for " << duration << " seconds." << endl;

	return NULL;
}

int main()
{
	const int NUM_ITTS = 5;

	for (int i = 0; i< NUM_ITTS; i++)
	{
		SleepTime((void*)&i);
	}

	return 0;
}
```

2. Write a program that has two counters. The main function should increment one of the counters up to 100. The threaded function should do the same to the other counter. Main should then print out both counters (hint: both counters will need to be local to main).

3. A limitation with using threads is that only a single argument can be passed into pthread_create. A way around this is to create a structure to hold more data. Create a structure that represents items from a supermarket. Each item should have an id, a name and a price (use a double). Show how a structure can be passed into a function on a new thread (pass it in and print out the contents).

4. You are creating a program that must print out the numbers 1 to 10000, the order is not important.

   Write a program that splits the work between 4 threads by each being responsible for a given sequence e.g. Thread 0 prints 1,5,9,13 etc, thread 1 prints 2, 6, 10, 14 etc.

   Change the program so that each thread processes blocks of 100 numbers and progress is recorded by a counter. The pseudocode for the function for each thread is as follows:

   While(count < 10000)

```
            myWork = count;
        count += 100;
        for(num = myWork:myWork+100)
            print(num);
```

where count is a shared variable.

Which method is faster? Are there any issues with the second version?