```cpp
/*
CPSC 457
Thread examples and pair examples
*/
// Example 1: Creating a thread using <thread>
#include <iostream>
#include <thread>
#include <unistd.h>

using namespace std;

void hello(){
    cout<<"Hi! I'm thread #" << this_thread::get_id() << endl;
}

int main(int argc, char const *argv[]){
    cout<<"About to create a thread \n";

    thread t1(hello);

    // wait for thread to finish
    t1.join();


    cout<<"Thread finished\n";
    return 0;
}

// ================================================================
// Example 2: Using multiple threads to sum and store result into an array
#include <iostream>
#include <thread>
#include <unistd.h>
#include <math.h>

//using namespace std;

long result[10];

void sum(long first, long end, int id)
{

//    std::cout << "Thread #" << id << std::endl;
    long s=0;
    for (long i=first; i<end; i++) s+=i;

    // Store result
    result[id] = s;
}

int main(int argc, char *argv[])
{
    std::cout << "Hello!" << std::endl;

    int nThreads = atoi(argv[1]);
    long number = 1000000000;

    long partitionSize = floor(number/nThreads);

    std::thread threadPool[nThreads];
```

```
    for (int i = 0; i < nThreads; i++)
    {
        // in case partitioning is not perfectly even
        if (i == nThreads - 1)
            threadPool[i] = std::thread(sum, i*partitionSize, number, i);
        else
            threadPool[i] = std::thread(sum, i*partitionSize,
(i+1)*partitionSize, i);

    }

    // wait for every thread to finish
    for (int i = 0; i < nThreads; i++)
    {
        threadPool[i].join();
    }

    // Add up result array
    long total = 0;
    for (int i = 0; i < nThreads; i++)
        total += result[i];

    std::cout<<"Sum = "<<total << std::endl;
    return 0;
}


// ================================================================
/*
Example 3: Create a vector of pairs
*/

#include <iostream>
#include <vector>
#include <utility>      // pair
#include <string>

using namespace std;


int main()
{
    vector <pair <string, int> > classList;

    pair <string, int> student1 ("Alice", 123456);
    pair <string, int> student2 ("Bob", 24127);
    pair <string, int> student3;

    // use make_pair function
    student3 = make_pair("Eve", 87654);

    classList.push_back(student1);
    classList.push_back(student2);
    classList.push_back(student3);

    cout << "Class list: " << endl;
    for (const auto &s: classList)
    {
        cout << "Student name: " << s.first << ", Student ID: " << s.second
<< endl;
```

```
119     }
120
121     return 0;
122 }
```