Concurrency in C++11

Arash

Threads

```
#include <iostream>
#include <thread>
using namespace std;

void func(int x, int y, int& z) {
    this_thread.sleep_for(chrono::seconds{5})
    cout << "Inside thread " << x << “, “ << y << “, “ << z << endl;
    x = 100; y = 200; z = 300;
}

int main() {
    int x = 2, y = 12; z = 102
    thread th{&func, x, ref(y), ref(z)}; // The first parameter can also be functor or lambda
    x++; y++; z++;
    cout << "Before join " << x << “, “ << y << “, “ << z<< endl;
    th.join();
    cout << "After join " << x << “, “ << y << “, “ << z << endl;
    return 0;
}
```

Mutex

```
template <typename T>
class container {
    mutex lock;
    vector<T> elements;
public:
    void add(T element){
      lock.lock();
      elements.push_back(element);
      lock.unlock();
    }
    template<typename... Args>
    void add(T first, Args... args) {
        add(first);
        add(args...);
    }
    void dump() {
        lock_guard<mutex> locker(_lock);
        for(auto e : _elements)
            cout << e << "  ";
        cout << endl;
    }
};
```

```
void func(container<int>& cont)
{
    cont.add(rand()%10, rand()%10, rand()%10);
}

int main()
{
    srand((unsigned int)time(0));
    container<int> cont;
    std::thread t1(func, ref(cont));
    std::thread t2(func, ref(cont));
    std::thread t3(func, ref(cont));
    t1.join();
    t2.join();
    t3.join();
    cont.dump();
    return 0;
}
```

Propagate exception between threads

```
mutex g_mutex;
vector<exception_ptr> g_exceptions;
void throw_function() {
   throw exception();
}
void func() {
   try {
      throw_function();
   }
   catch(...) {
      lock_guard<mutex> lock(g_mutex);
g_exceptions.push_back(current_exception());
   }
}
```

```
int main() {
   g_exceptions.clear();
   thread t(func);
   t.join();
   for(auto& e : g_exceptions) {
      try {
         if(e != nullptr) {
            rethrow_exception(e);
         }
      }
      catch(const exception& e) {
         cout << e.what() << endl;
      }
   }
   return 0;
}
```

Condition Variable

```
const int size = 10;
mutex mutex;
condition_variable condvar;

struct Data {
    int d;
    int error;
};

queue<Data> messageQ;

void Producer()
{
    int i = 0;
    while(++i<=size) {
        int d = rand()%10;
        int error = (i==size)?1:0;
        Data data {d, error};
        this_thread::sleep_for(milliseconds(rand()%500));
        lock_guard<std::mutex> guard(mutex);
        cout << "Producing message: " << data.d << endl;
        messageQ.push(data);
        condvar.notify_one();
    }
}
void Consumer()
{
    while(1) {
        this_thread::sleep_for(milliseconds(rand()%500));
        unique_lock<std::mutex> ulock(mutex);
        condvar.wait(ulock, [] {return !messageQ.empty(); });

        Data data = messageQ.front();
        cout << "Consuming message: " << data.d  << endl;
        messageQ.pop();
        if (data.error)
            break;
    }
}
```

```
int main() {
   std::thread t1 {Producer};

   std::thread t2 {Consumer};

   t1.join();

   t2.join();
}
```

Async

Future and Promises

Resources

- [1] http://www.codeproject.com/Articles/598695/Cplusplus-threads-locks-and-condition-variables