Week 7

Exercise 55

../55/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include <vector>
#include <iostream>
#include <chrono>
#include <thread>
#include <algorithm>

void calcPrimes(size_t noPrimes, bool &calcDone, vector<size_t> &primes);
void dotting(size_t seconds, bool &calcDone);
```

../55/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  if (argc != 2)  // Conditional exit
  {
    cerr << "Please specify the number of primes to compute \n";
    return 0;
  }

  bool calcDone = false;      // Done calculating?
  // Only written to and read by one thread at a time: no atomic or mutex needed

  vector<size_t> primes{ 2 }; // Vector of primes, primed with '2'

  auto startChrono = chrono::system_clock::now();
  time_t start = chrono::system_clock::to_time_t(startChrono);
  // Starting time

  size_t noPrimes = stoi(argv[1]);
  // Number of primes to calculate

  thread dottingThread(dotting, 1, ref(calcDone));
  // Start the dotting
  thread primesThread(calcPrimes, noPrimes, ref(calcDone), ref(primes));
  // Start the calculating

  primesThread.join();
  dottingThread.join();
  // Waiting for both to be done

  cout << '\n';
  for (auto el: primes)
    cout << el << ' ';
  // Output primes

  auto endChrono = chrono::system_clock::now();
  time_t end = chrono::system_clock::to_time_t(endChrono);
  // End time

  chrono::duration<double> elapsed_seconds = endChrono - startChrono;
  // Duration calculation

  cout << "\nStarting time: " << ctime(&start)
```

```
45        << "Ending time:    " << ctime(&end)
46        << "Computation of " << stoi(argv[1]) << " primes took "
47                             << elapsed_seconds.count() << " seconds";
48    // Output timing
49  }
```

../55/calcPrimes.cc

```
1   #include "main.ih"
2
3   void calcPrimes(size_t noPrimes, bool &calcDone, vector<size_t> &primes)
4   {
5     size_t next = 3;
6
7     while (primes.size() < noPrimes)
8     {
9       bool isPrime = none_of(
10                              primes.begin(),
11                              primes.end(),
12                              [&](auto el)
13                              {
14                                return next % el == 0;
15                              }
16                             );
17      if (isPrime)
18        primes.push_back(next);
19      ++next;
20    }
21    calcDone = true;
22  }
```

../55/dotting.cc

```
1   #include "main.ih"
2
3   void dotting(size_t seconds, bool &calcDone)
4   {
5     while (!calcDone)
6     {
7       std::this_thread::sleep_for(std::chrono::seconds(seconds));
8       cerr << '.';
9     }
10  }
```

Exercise 56

../56/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include <iostream>
#include <chrono>
```

../56/main.cc

```
#include "main.ih"

#include <cmath>

int main(int argc, char const **argv)
{
  if (argc != 3)                          // Conditional exit
  {
    cout << "Please pass two arguments.";
    return 1;
  }

  chrono::hours   inputHours(stoi(argv[1]));    // Arg 1, input hours
  chrono::seconds inputSeconds(stoi(argv[2]));  // Arg 2, input seconds

  chrono::minutes outputMinutesH = inputHours;
  // Conversion from hours to minutes: more granular, no cast
  chrono::minutes outputMinutesS =
    chrono::duration_cast<chrono::minutes>(inputSeconds);
  // Conversion from seconds to minutes: less granular, loses precision, needs
  // cast

  cout << inputHours.count()   << " hours equals "
                               << outputMinutesH.count() << " minutes \n"
       << inputSeconds.count() << " seconds roughly equals "
                               << outputMinutesS.count() << " minutes";
  // Output
}
```

Exercise 57

../57/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include <string>   // Input arguments
#include <iostream> // Output
#include <chrono>   // chrono:: facilities
#include <iomanip>  // put_time
```

../57/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  if (argc != 2)                          // Conditional exit
  {
    cerr << "Please pass an argument";
    return 1;
  }

  string argvString = argv[1];  // Offset string (for s, m, h)
  int offset = stoi(argv[1]);   // Offset integer, int for possible negatives

  auto adjClock = chrono::system_clock::now();                  // Curr. time
  time_t adjClockT = chrono::system_clock::to_time_t(adjClock); // Same, time_t

  cout << "Current time      : " << put_time(localtime(&adjClockT), "%c") << '\n'
       << "Current time (GMT): " << put_time(gmtime(&adjClockT), "%c") << "\n"
       << "Adjusted time     : ";
  // Basic output of current time in local timezone and GMT

  switch (argvString.back())
  {
    case 's':
      adjClockT =
        chrono::system_clock::to_time_t(adjClock + chrono::seconds{offset});
      cout << put_time(localtime(&adjClockT), "%c");
      break;
    case 'm':
      adjClockT =
        chrono::system_clock::to_time_t(adjClock + chrono::minutes{offset});
      cout << put_time(localtime(&adjClockT), "%c");
      break;
    case 'h':
      adjClockT =
        chrono::system_clock::to_time_t(adjClock + chrono::hours{offset});
      cout << put_time(localtime(&adjClockT), "%c");
      break;
    default:
      cout << "Invalid time offset."; // Invalid input
      return 1;
  }
  // Switch based on last letter of input string (s, m, h): determines offset
  // for adjusted time
}
```

Exercise 58

../58/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include "handler/handler.h"

#include <iostream>
#include <thread>
#include <fstream>

void execShift(Handler &myHandler, std::ostream &out, string const &text);
void defShift(ostream &out, string const &text);
```

../58/main.cc

```
#include "main.ih"


int main(int argc, char const **argv)
{
  if (argc != 3)                  // Conditional exit
  {
    cerr << "Invalid argument.";
    return 1;
  }

  string inputString = argv[2];   // Word to be shifted

  Handler wordHandler;            // Define Handler
  ofstream myfile (argv[1]);      // Open file
  thread thread1(execShift, ref(wordHandler), ref(myfile), ref(inputString));
                                  // Pass Handler to thread1, execute shift
  thread1.join();                 // Wait for thread1 to finish
  myfile.close();                 // Close the file

  myfile.open(argv[1]);           // Reopen the file
  thread thread2(defShift, ref(myfile), ref(inputString));
                                  // Pass the stream and string to thread2,
                                  // which itself creates a Handler
  thread2.join();                 // Wait for thread2 to finish
}
```

../58/defShift.cc

```
#include "main.ih"

void defShift(ostream &out, string const &text)
{
  Handler myHandler;
  myHandler.shift(out, text);
}
// Defines Handler in-thread, then asks it to perform the shift function
```

../58/execShift.cc

```
#include "main.ih"

void execShift(Handler &myHandler, ostream &out, string const &text)
{
  myHandler.shift(out, text);
```

```
6 }
7 // Gets passed a thread, which is asked to call shift
```

../58/handler/handler.h

```
1  #ifndef INCLUDED_HANDLER_
2  #define INCLUDED_HANDLER_
3
4  #include <iosfwd>
5  #include <string>
6
7  class Handler
8  {
9    public:
10     void shift(std::ostream &out, std::string const &text);
11 };
12
13 #endif
```

../58/handler/handler.ih

```
1  #include "handler.h"
2
3  using namespace std;
4
5  #include <iostream>
```

../58/handler/shift.cc

```
1  #include "handler.ih"
2
3  void Handler::shift(ostream &out, string const &text)
4  {
5    size_t counter = 0;                  // Counter for modulo
6    size_t textLength = text.length();   // Length of string to avoid IRE
7
8    for (size_t idx = 0; idx != textLength; ++idx)  // Loop length of string
9    {
10     for (size_t idx2 = 0; idx2 != textLength; ++idx2) // Loop again
11     {
12       out << text[counter % textLength];  // Output character based on counter
13       ++counter;                          // Increment counter
14     }
15     out << '\n';                          // New line, new iteration
16     ++counter;                            // Increment again after each 'word'
17                                           // so that the shift occurs
18   }
19
20   // Second approach:
21   // string outputString = text;
22   // out << outputString;
23   // for (size_t idx = 0; idx != text.length() - 1; ++idx)
24   // {
25   //   outString += outString.front();
26   //   outString.erase(0, 1);
27   //   out << outString << '\n';
28   // }
29 }
```

Exercise 59

../59–E/main.ih

```
#include "storage/storage.h"

#include <thread>
#include <iostream>

using namespace std;

void addlines(Storage &warehouse, istream &input);
```

../59–E/main.cc

```
#include "main.ih"

int main(int argc, char const *argv[])
{
  Storage warehouse(argv[1]);
  // Define warehouse, given filename

  thread addTread(&addlines, ref(warehouse), ref(cin));
  // Add lines to the queue
  thread runThread(&Storage::run, ref(warehouse));
  // Process those lines

  addTread.join();
  runThread.join();
}
```

../59–E/addlines.cc

```
#include "main.ih"

void addlines(Storage &warehouse, istream &input)
{
  string inputString;
  while (cin >> inputString)      // While there is still user input
    warehouse.push(inputString);  // Push that input to the queue

  warehouse.finished();           // When input is done, signal that
}
```

../59–E/storage/storage.h

```
#ifndef INCLUDED_STORAGE_
#define INCLUDED_STORAGE_

#include <queue>
#include <string>
#include <mutex>

class Storage
{
  std::mutex              d_mutex;
  std::queue<std::string> d_queue;
  bool                    d_finished = false;
  std::string             d_outputFile;

  public:
    Storage(std::string outputFile);  // Constructor based on filename

    bool          empty();
```

```
19      std::string  next();
20      void         push(std::string const line);
21      std::string &front();
22      void         finished();
23      void         run();
24  };
25
26  #endif
```

../59–E/storage/storage.ih

```
1   #include "storage.h"
2
3   #include <fstream>
4   #include <chrono>
5   #include <thread>
6
7   using namespace std;
```

../59–E/storage/c_storage.cc

```
1   #include "storage.ih"
2
3   Storage::Storage(string outputFile)
4     : d_outputFile(outputFile)
5   {
6   }
```

../59–E/storage/empty.cc

```
1   #include "storage.ih"
2
3   bool Storage::empty()
4   {
5     return d_queue.empty();
6   }
```

../59–E/storage/finished.cc

```
1   #include "storage.ih"
2
3   void Storage::finished()
4   {
5     d_finished = true;
6   }
```

../59–E/storage/front.cc

```
1   #include "storage.ih"
2
3   string &Storage::front()
4   {
5     return d_queue.front();
6   }
```

../59–E/storage/next.cc

```
1   #include "storage.ih"
2
3   string Storage::next()
4   {
5     lock_guard<std::mutex> mx(d_mutex); // Lock queue
6     string front = d_queue.front();     // Get element from queue
```

```
7     d_queue.pop();                        // Remove that element
8     return front;                         // Return it
9   }
```

../59–E/storage/push.cc

```
1   #include "storage.ih"
2
3   void Storage::push(string const line)
4   {
5     lock_guard<mutex> lk(d_mutex);  // Lock queue
6     d_queue.push(line);             // Add new element to the queue
7   }
```

../59–E/storage/run.cc

```
1   #include "storage.ih"
2
3   void Storage::run()
4   {
5     ofstream outputStream(d_outputFile);  // Output file
6     while (!d_finished || !empty())       // While queue is not empty, or cin
7     {                                     // is still writing to the queue
8       if (!empty())                       // If the queue is not empty
9         outputStream << next() << '\n';   // Output the element from the queue
10
11      this_thread::sleep_for(1s);         // Sleep
12    }
13  }
```