

## Week 7

### Exercise 55

../55/main.ih

```
1  #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3  using namespace std;
4
5  #include <vector>
6  #include <iostream>
7  #include <chrono>
8  #include <thread>
9  #include <algorithm>
10
11 void calcPrimes(size_t noPrimes, bool &calcDone, vector<size_t> &primes);
12 void dotting(size_t seconds, bool &calcDone);
```

../55/main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char const **argv)
4  {
5      if (argc != 2) // Conditional exit
6      {
7          cerr << "Please specify the number of primes to compute \n";
8          return 0;
9      }
10
11     bool calcDone = false; // Done calculating?
12     // Only written to and read by one thread at a time: no atomic or mutex needed
13
14     vector<size_t> primes{ 2 }; // Vector of primes, primed with '2'
15
16     auto startChrono = chrono::system_clock::now();
17     time_t start = chrono::system_clock::to_time_t(startChrono);
18     // Starting time
19
20     size_t noPrimes = stoi(argv[1]);
21     // Number of primes to calculate
22
23     thread dottingThread(dotting, 1, ref(calcDone));
24     // Start the dotting
25     thread primesThread(calcPrimes, noPrimes, ref(calcDone), ref(primes));
26     // Start the calculating
27
28     primesThread.join();
29     dottingThread.join();
30     // Waiting for both to be done
31
32     cout << '\n';
33     for (auto el: primes)
34         cout << el << ' ';
35     // Output primes
36
37     auto endChrono = chrono::system_clock::now();
38     time_t end = chrono::system_clock::to_time_t(endChrono);
39     // End time
40
41     chrono::duration<double> elapsed_seconds = endChrono - startChrono;
42     // Duration calculation
43
44     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

```
1  #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3  using namespace std;
4
5  #include <iostream>
6  #include <chrono>
```

../56/main.cc

```
1  #include "main.ih"
2
3  #include <cmath>
4
5  int main(int argc, char const **argv)
6  {
7      if (argc != 3)                                // Conditional exit
8      {
9          cout << "Please pass two arguments.";
10         return 1;
11     }
12
13     chrono::hours   inputHours(stoi(argv[1]));      // Arg 1, input hours
14     chrono::seconds inputSeconds(stoi(argv[2]));    // Arg 2, input seconds
15
16     chrono::minutes outputMinutesH = inputHours;
17     // Conversion from hours to minutes: more granular, no cast
18     chrono::minutes outputMinutesS =
19         chrono::duration_cast<chrono::minutes>(inputSeconds);
20     // Conversion from seconds to minutes: less granular, loses precision, needs
21     // cast
22
23     cout << inputHours.count() << " hours equals "
24          << outputMinutesH.count() << " minutes \n"
25          << inputSeconds.count() << " seconds roughly equals "
26          << outputMinutesS.count() << " minutes";
27     // Output
28 }
```

## Exercise 57

../57/main.ih

```
1 #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3 using namespace std;
4
5 #include <string>    // Input arguments
6 #include <iostream>  // Output
7 #include <chrono>    // chrono:: facilities
8 #include <iomanip>    // put_time
```

../57/main.cc

```
1 #include "main.ih"
2
3 int main(int argc, char const **argv)
4 {
5     if (argc != 2)                // Conditional exit
6     {
7         cerr << "Please pass an argument";
8         return 1;
9     }
10
11     string argvString = argv[1];  // Offset string (for s, m, h)
12     int offset = stoi(argv[1]);   // Offset integer, int for possible negatives
13
14     auto adjClock = chrono::system_clock::now();           // Curr. time
15     time_t adjClockT = chrono::system_clock::to_time_t(adjClock); // Same, time_t
16
17     cout << "Current time      : " << put_time(localtime(&adjClockT), "%c") << '\n'
18          << "Current time (GMT): " << put_time(gmtime(&adjClockT), "%c") << "\n"
19          << "Adjusted time      : ";
20     // Basic output of current time in local timezone and GMT
21
22     switch (argvString.back())
23     {
24         case 's':
25             adjClockT =
26                 chrono::system_clock::to_time_t(adjClock + chrono::seconds{offset});
27             cout << put_time(localtime(&adjClockT), "%c");
28             break;
29         case 'm':
30             adjClockT =
31                 chrono::system_clock::to_time_t(adjClock + chrono::minutes{offset});
32             cout << put_time(localtime(&adjClockT), "%c");
33             break;
34         case 'h':
35             adjClockT =
36                 chrono::system_clock::to_time_t(adjClock + chrono::hours{offset});
37             cout << put_time(localtime(&adjClockT), "%c");
38             break;
39         default:
40             cout << "Invalid time offset."; // Invalid input
41             return 1;
42     }
43     // Switch based on last letter of input string (s, m, h): determines offset
44     // for adjusted time
45 }
```

## Exercise 58

../58/main.ih

```
1  #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3  using namespace std;
4
5  #include "handler/handler.h"
6
7  #include <iostream>
8  #include <thread>
9  #include <fstream>
10
11 void execShift(Handler &myHandler, std::ostream &out, string const &text);
12 void defShift(ostream &out, string const &text);
```

../58/main.cc

```
1  #include "main.ih"
2
3
4  int main(int argc, char const **argv)
5  {
6      if (argc != 3)                // Conditional exit
7      {
8          cerr << "Invalid argument.";
9          return 1;
10     }
11
12     string inputString = argv[2];  // Word to be shifted
13
14     Handler wordHandler;           // Define Handler
15     ofstream myfile (argv[1]);     // Open file
16     thread thread1(execShift, ref(wordHandler), ref(myfile), ref(inputString));
17                                     // Pass Handler to thread1, execute shift
18     thread1.join();               // Wait for thread1 to finish
19     myfile.close();               // Close the file
20
21     myfile.open(argv[1]);          // Reopen the file
22     thread thread2(defShift, ref(myfile), ref(inputString));
23                                     // Pass the stream and string to thread2,
24                                     // which itself creates a Handler
25     thread2.join();               // Wait for thread2 to finish
26 }
```

../58/defShift.cc

```
1  #include "main.ih"
2
3  void defShift(ostream &out, string const &text)
4  {
5      Handler myHandler;
6      myHandler.shift(out, text);
7  }
8  // Defines Handler in-thread, then asks it to perform the shift function
```

../58/execShift.cc

```
1  #include "main.ih"
2
3  void execShift(Handler &myHandler, ostream &out, string const &text)
4  {
5      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     //     outputString += outputString.front();
26     //     outputString.erase(0, 1);
27     //     out << outputString << '\n';
28     // }
29 }
```

## Exercise 59

## Exercise 60