Week 7: II

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});
      break;
    case 'm':
      adjClockT =
        chrono::system_clock::to_time_t(adjClock + chrono::minutes{offset});
      break;
    case 'h':
      adjClockT =
        chrono::system_clock::to_time_t(adjClock + chrono::hours{offset});
      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
  cout << put_time(localtime(&adjClockT), "%c");
}
```

Exercise 60

../60–4/main.ih

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

#include "client/client.ih"
#include "warehouse/warehouse.ih"

#include <iostream>
#include <string>
#include <thread>



using namespace std;
```

../60–4/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  Warehouse warehouse;

  vector<Client> clients;

  for (size_t idx = 1; idx < argc; ++idx)
    clients.emplace_back( warehouse, argv[idx]);
//adding clients to the clients vector

  vector<thread> threads;

  for (auto &client: clients)
    threads.emplace_back(&Client::printProduct, ref(client));

  thread addThread(&Warehouse::addlines, ref(warehouse));
  //thread that adds lines to the queue

//adding a thread that takes lines from the queue in the warehouse and prints
//it in the file corresponding to the client.

  addThread.join();  //joining threads

  for (auto &it: threads)
    it.join();

  for (auto client: clients)  //printing nr of lines per file
    cout << client.size() << '\n';

}
```

../60–4/client/client.h

```
#ifndef INCLUDED_CLIENT_
#define INCLUDED_CLIENT_

#include "../warehouse/warehouse.ih"

class Client
{
  Warehouse &d_warehouse;
  std::string d_outputFile;
  size_t d_nrlines = 0;

  public:
    Client(Warehouse &warehouse, std::string const &outputFile);

    size_t size();        //returns nr of lines printed
    void printProduct(); //prints strings to file
  private:
};

#endif
```

../60–4/client/client.ih

```
#include "client.h"
#include <fstream>
#include <iostream>
#include <chrono>
#include <thread>


using namespace std;
```

../60–4/client/c_client.cc

```
#include "client.ih"

Client::Client(Warehouse &warehouse, string const &outputFile)
: d_warehouse(warehouse),
  d_outputFile(outputFile)
{

}
```

../60–4/client/printProduct.cc

```
#include "client.ih"

void Client::printProduct()
{
  ofstream outputStream(d_outputFile);

  while(!(d_warehouse.empty() && d_warehouse.isitfinished()))
  {
    bool printit = true;
    //printit is used to distinguish between an empty string that should be
    //printed as it is an empty string in the warehouse and an empty string
    //that is returned because there is nothing in the warehouse. In the
    //latter case printit is set to false and it wont be printed.
    string tmp = d_warehouse.getProduct(printit);

    if(printit)
    {
      outputStream << tmp << '\n';
      ++d_nrlines;
    }
  }
}
```

../60–4/client/size.cc

```
#include "client.ih"

size_t Client::size()
{
  return d_nrlines;
}
```

../60–4/warehouse/warehouse.h

```cpp
#ifndef INCLUDED_WAREHOUSE_
#define INCLUDED_WAREHOUSE_

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

class Warehouse
{
  std::queue<std::string> d_queue;
  std::mutex wMutex;
  std::condition_variable condition;
  bool d_finished = false;

  public:
    Warehouse();

    std::string &front(); //returns string that has been in queue the longest

    bool empty(); //checks whether queue is empty

    std::string getProduct(bool &printit);
                              //used by clients to retrieve a string from
                              // the queue

    bool isitfinished(); //checks if there is more input to come

    void addlines(); //processes input to queue and calls finished when there
                     //is no more input

  private:
    std::string next(); //removes and returns string from queue
    void addProduct(std::string const &line); //adds a string to the queue
    void finished(); //sets d_finished to true and notifies all waiting

};

#endif
```

../60–4/warehouse/warehouse.ih

```cpp
#include "warehouse.h"
#include <iostream>


using namespace std;
```

../60–4/warehouse/addLines.cc

```cpp
#include "warehouse.ih"

void Warehouse::addlines()
{
  string inputString;
  while (getline(cin, inputString))       // While there is still user input
    addProduct(inputString);       // Push that input to the queue

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

../60–4/warehouse/addProduct.cc

```cpp
#include "warehouse.ih"
```

```

void Warehouse::addProduct(string const &line)
{
  lock_guard<mutex> lk(wMutex);
  d_queue.push(line);

  if (d_queue.size() == 1)
    condition.notify_all(); //notify waiting clients a string is available

}
```

../60–4/warehouse/c_warehouse.cc

```
#include "warehouse.ih"

Warehouse::Warehouse()
//:
{
}
```

../60–4/warehouse/empty.cc

```
#include "warehouse.ih"

bool Warehouse::empty()
{
  return d_queue.empty();
}
```

../60–4/warehouse/finished.cc

```
#include "warehouse.ih"

void Warehouse::finished()
{
  d_finished = true;
  condition.notify_all(); //notify waiting processes there will be no more
                          //products so they should stop waiting for one.
}
```

../60–4/warehouse/front.cc

```
#include "warehouse.ih"

string &Warehouse::front()
{
  return d_queue.front();
}
```

../60–4/warehouse/getProduct.cc

```
#include "warehouse.ih"

string Warehouse::getProduct(bool &printit)
{
  unique_lock<mutex> ul(wMutex);
  while (empty() && !d_finished)
    condition.wait(ul);

  if (!empty())
    return next();

  printit = false;
  return {};  //return empty string, which can happen if no more strings
```

```
                    //are going to be available
}
```

../60–4/warehouse/isitfinished.cc

```
#include "warehouse.ih"

bool Warehouse::isitfinished()
{
  return d_finished;
}
```

../60–4/warehouse/next.cc

```
#include "warehouse.ih"

string Warehouse::next()
{
  string front = d_queue.front();      // Get element from queue
  d_queue.pop();                       // Remove that element

  return front;                        // Return it
}
```