**MSCF Programming Prep**

**Homework 1**

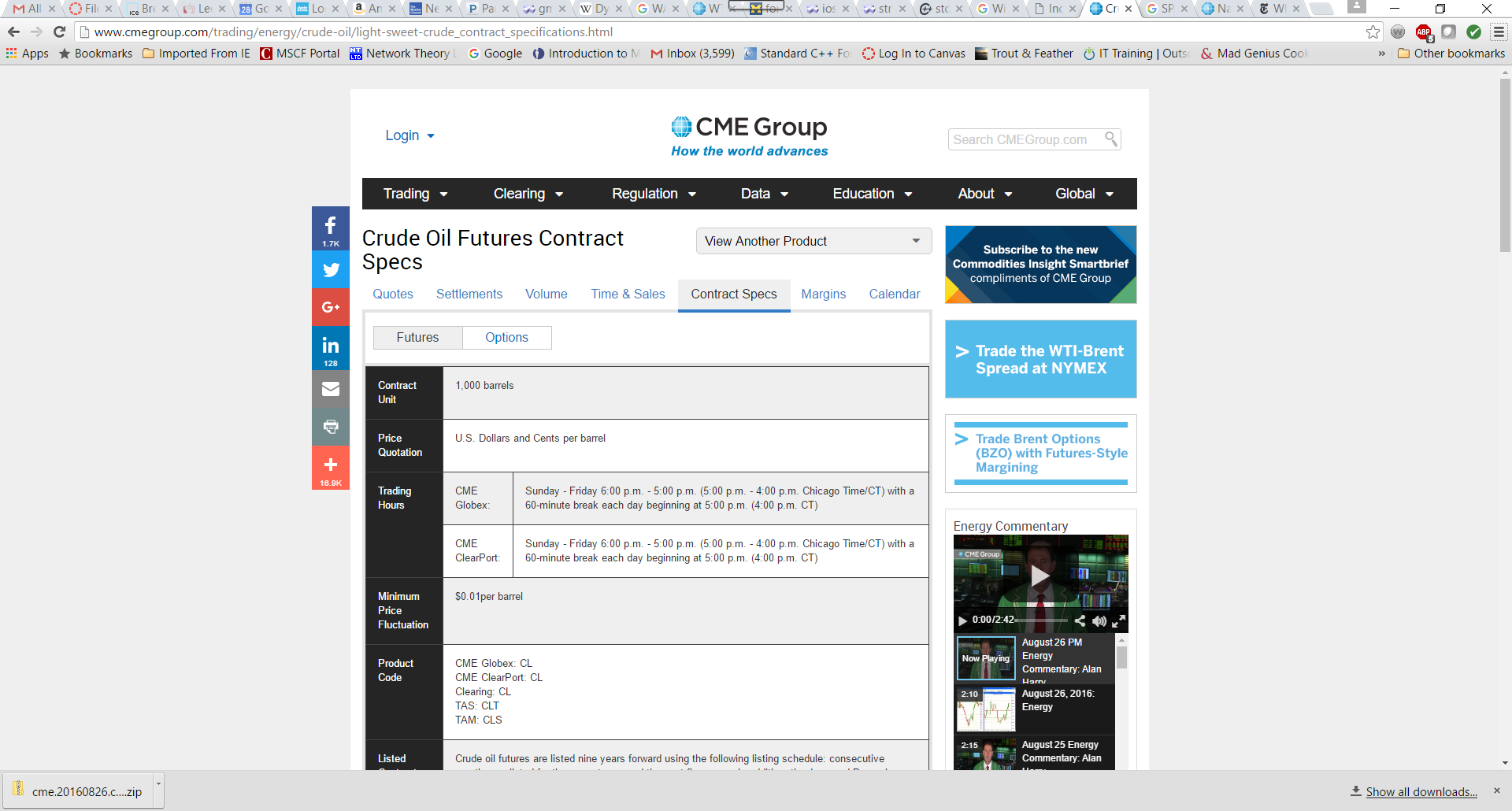
***Due At 11:59 pm Monday, Aug. 6, 2018***

1. (70 points) Commodity futures and option contracts of many kinds are traded on NYMEX, owned by CME Group. Each evening of each trading day, sometime between about 6:00 pm and 8:00 pm Central Time, a SPAN (Standard Portfolio Analysis of Risk) file is posted to <ftp://ftp.cmegroup.com/pub/span/data/cme> containing information about the day’s trading. For a given day, the name of this file is cme.*YYYYMMDD*.c.pa2.zip, where *YYYYMMDDD* is the 8-digit year, month, and day of the file. Files for months prior to the current month are moved into the [/pub/span/data/cme/201](ftp://ftp.cmegroup.com/pub/span/data/cme/2016)8 subdirectory.

Download the zipped SPAN file for Friday, July 27, 2018, cme.20180727.c.pa2.zip. Unzip, then display this SPAN file. You will see that it is an enormous text file with its own unique format, unfortunately *not* something simple and convenient like CSV or XML or JSON.

The *settlement prices* contained in the SPAN file are used to *mark to market* each trader’s account, so that gains/losses can be credited/debited each day to reduce the risk of counterparty default. Your job is to extract these settlement prices, as well as contract expiration dates (last trading dates), for two of the most heavily traded energy contracts: West Texas Intermediate (WTI) Crude Oil, and Henry Hub Natural Gas.

To learn about WTI Crude Oil futures contract details, check out: <http://www.cmegroup.com/trading/energy/crude-oil/light-sweet-crude_contract_specifications.html>



Notice that the CME Globex Product Code is CL; you will need this for scanning the SPAN file. Using other tabs at the top of this page, you can see current quotes, recent settlements, volume, etc. If you click the Options button, just to the right of the Futures button near the upper left, you will see information about options contracts based on the underlying futures contracts. There are about two dozen different types of option contracts for this underlying; we are interested in the American Options. When you look at the contract specifications, you will discover that its Product Code is LO.

In the View Anther Product menu at upper right, select Natural Gas (Henry Hub) Physical Futures to learn analogous information about natural gas futures contracts and the corresponding American option contracts.

Write a C++ program named **hw1.1.cpp** that reads **cme.20170731.c.pa2** as its input file, and produces **CL\_and\_NG\_expirations\_and\_settlements.txt** as its output file. The output should be in exactly this form:

Futures Contract Contract Futures Options Options

Code Month Type Exp Date Code Exp Date

------- -------- -------- -------- ------- --------

CL 2018-10 Fut 2018-09-20

CL 2018-11 Fut 2018-10-20

*… and so forth, through contract month 2020-12 …*

CL 2018-10 Opt LO 2018-09-17

CL 2018-11 Opt LO 2018-10-17

*… and so forth, through contract month 2020-12 …*

NG 2018-10 Fut 2018-09-26

NG 2018-11 Fut 2018-10-29

*… and so forth, through contract month 2020-12 …*

NG 2018-10 Opt ON 2018-09-25

NG 2018-11 Opt ON 2018-10-26

*… and so forth, through contract month 2020-12 …*

Futures Contract Contract Strike Settlement

Code Month Type Price Price

------- -------- -------- ------ ----------

CL 2018-10 Fut 67.73

CL 2018-11 Fut 67.36

*… and so forth, through contract month 2020-12 …*

CL 2018-10 Call 5.50 62.48

CL 2018-10 Put 5.50 0.01

CL 2018-10 Call 6.00 61.98

CL 2018-10 Put 6.00 0.01

*… and so forth, through contract month 2020-12 …*

NG 2018-10 Fut 2.800

NG 2018-11 Fut 2.848

*… and so forth, through contract month 2020-12 …*

NG 2018-10 Call 0.650 2.152

NG 2018-10 Put 0.650 0.001

NG 2018-10 Call 0.700 2.102

NG 2018-10 Put 0.700 0.001

*… and so forth, through contract month 2020-12 …*

***Do not try to create a better output format***: it needs to be very easy for us to compare your output to our solution output, and to other students’ outputs. We will take off points if your output format varies too much from what is shown above. Our output format takes into account the order in which records appear in the SPAN file, so you don’t have to remember or accumulate too much information as you go.

***Do not*** include contract months earlier than 2018-10, or later than 2020-12.

Since there are many, many strike prices for options on futures contracts, the output file is going to be very long, but not nearly as long as the SPAN file itself.

Fortunately, there is documentation online that describes the contents of CME SPAN files. If you Google for “cme span pa2 file format” you will find a page named “Risk Parameter File Layouts for the Positional Formats – SPAN…”. You will want to look at Type “B” Records, Expanded Format, and Type “8” Records, Expanded Format, to learn how to obtain the contract name, type, month, expiration date, strike, and settlement prices that you need.

A few hints:

(a) Notice that the documentation counts character column positions from 1, whereas in your code you will need to count character positions from 0 for **substr** or other purposes

(b) Check the contract specifications to discover the number of decimal places you should display for prices of different commodity futures and options contracts.

(c) You will discover that the documentation is not quite perfect, but you should be able to figure out any problem(s) you encounter. (Sub-hint: talk to others.)

(d) Approach the program in stages: first, make sure you can write a program that simply copies the SPAN file to the output file; next, copy the type B and type 8 records from the SPAN file to the output file; next, copy the type B and type 8 CL records; next, the type B and type 8 CL and NG records; and so forth, making definite steady progress with each revision. As your coding skills improve, you can do two or three or four things in each revision step. Eventually, you will find that you can write dozens of lines of code encompassing many different tasks and goals, and it will compile and work the first time! But maybe not every time.

(e) You can use any C++ facilities that you know, but you should not need anything other than what was discussed in the Day 1 through Day 4 Lecture Notes and the “Various Things” document.

(f) Remember the Discussion board and **jostlund@andrew.cmu.edu**.

At the top of your source code file, put in comments including this information:

**// Programming Prep, Fall 2018**

**// Homework 1.1**

**// File: hw1.1.cpp**

**// Authors: *the names of the two homework partners***

**// Description:** *[what does your program do? Be clear, complete and concise.]*

Comment your code as you deem necessary: major steps, or tricky logic. *Don’t* include gratuitous comments, like:

**int main()** // this is the main function, that takes no parameters and returns int

Choose meaningful, so-called self-documenting variable names. For example:

**double strike\_price; /\* this is good! No comment necessary \*/**

**double sp; // strike price /\* this is lazy and stupid, even with comment \*/**

1. (15 points) We have used the C++ *fundamental data types* **int** and **double**. Each of these is able to store values in an implementation-defined range. On 64-bit systems, such as our laptops, the typical range of values for **int** is roughly +/-2 billion. What happens when you go outside this range?
   1. Write a program named **hw1.2.cpp** that has one **int** variable, and assign the value **1000000000** (1 billion) to this variable. Display the value using **cout**.
   2. Now, triple the variable’s value and display the new result. Does your program crash? What is going on? Explain.
   3. Triple the value again and display. Explain.
   4. In your program, add a **double** variable with the initial value **1.0**. Display the value using **cout**. How many times can you increase this variable’s value by a factor of 2 before you get a “crazy” result?
   5. Have a look at **cplusplus.com** to find out about the **climits** standard header and the **INT\_MAX** constant. Add another **int** variable to your program with initial value **INT\_MAX**, then display the variable’s value. What is **INT\_MAX** on your laptop?
   6. Add **1** to the variable. What is its value now? Explain.

Add the necessary course name/Homework 1.2/File/Authors/Description comment block at the top of your source code file. Add comments *in your source code* giving your results and explanations for questions (a) through (f).

1. (15 points) Constants like **INT\_MAX** in **<climits>** and **DBL\_MAX** in **<cfloat>** give minimum and maximum values of your compiler’s integer and floating point fundamental data types. The **sizeof** *operator* applied to a data type tells how many bytes of memory a variable of that type occupies. For example, on your laptop **sizeof (int)** is most likely 4, and **sizeof (double)** is most likely 8. *Notice* that the type name is enclosed in parentheses: **sizeof (***type\_name***)**. You can also use **sizeof** to find out the size of a variable or a constant; in this case, the parentheses are not required:

**int i = 12;**

**cout << "sizeof (int): " << sizeof (int) << "\n";**

**cout << "sizeof i: " << sizeof i << "\n";**

**cout << "sizeof 12: " << sizeof 12 << "\n";**

**cout << "sizeof 12.3: " << sizeof 12.3**

**<< "\n"; // 12.3 has type double**

Even though we are not going to be using all of the C++ fundamental data types in this course, you should nevertheless have some knowledge of their sizes and ranges on any machine you are programming on.

Write a program named **hw1.3.cpp** that displays the sizes and ranges of values on your system of all the fundamental data types (except **void** and **nullptr**):

**bool char signed char unsigned char**

**short int long long long**

**unsigned short unsigned unsigned long unsigned long long**

**float double long double**

(Although **signed char** and **unsigned char** are the same size as a **char**, these types are really intended as very small signed and unsigned integer data types, respectively. At one point during the standardization process, the committee gave consideration to calling these **short short int** and **unsigned short short int**, in symmetry with **long long int** and **unsigned long long int**.)

Comment your code as you deem necessary, and add the needed comment block at the top of your source code file.

***And Finally***

Submit your homework electronically to Canvas. You and your partner should submit a single **.zip** file, containing the three source code files you wrote for the three parts of this homework.