IGME-451 System Concepts for Games and Media
Homework 3 – Regular Expressions
Due: April 1, 2019 at 12 NOON

Objective:

Learn to use regular expressions and dictionaries to parse, store, and query contents from a configuration file.

Configuration file description:

```
# This is a comment line. Comments continue to the
# end of line and should not be stored
# Comments can occur anywhere in a line and continue to the end

# whitespace and blank lines are ignored

# Keys are scoped by their section subsection
# redefinition of a key will eliminate the prior entry (replacement)

[section]       # Square brackets denote a section.
                # Key/value pairs are organized into sections.
                # You can use any alphanum to define the name of a
                # section.  Whitespaces are illegal.

[section:subsection]  # colon defines a subsection.
                      # This allows hierarchical creation of configs

# NOTE: ALL key values must be in a section or subsection
# There can be zero or more key value pairs per section
# Types should be enumerated within your functions
# e.g.: CONFIG_BOOLEAN_T, CONFIG_INT_T, CONFIG_FLOAT_T,
#       CONFIG_STRING_T, CONFIG_LIST_T, CONFIG_DEFAULT_T

key=value       # Set the key to a specific value
                # Value types can be as follows
                # Booleans (true, false caps insensitive)
                # integers (decimal without .)
                # floats  (decimal with . AND ending in f 1.0f
                # strings (defined by double quotes “hello”)
                # NOTE: Whitespaces are defined in quotes “as-is”

key={item1;item2}
                # Create a list associated with a key
                # List must be 1 or more elements long
                # Items must be of the appropriate value types
                # Any value type listed in key=value is valid!

key=            # Tell the entry to set to the game engine default
```

Description:

You will write a C++ program that uses std::regex and associated functions/classes/methods to parse a config file as described above. You should create reasonable data structures using dictionaries (or appropriate hashtable constructs) to represent sections/subsections, keys, values. Each key/value must support type information. You will add queries that allow you to do the following:

ListAllSections – Return a list of all sections from the config file
ListNamedSection – Returns a named section it it exists
ListSubsections – List subsections for a named section
ListAllEntries – List all key/value pairs for a section/subsection
GetEntry – Gets an entry by key from a named section/subsection
GetKey – Returns the key part of the entry
GetValue – Returns the value part of the entry
GetType – Returns the type of the entry

During the parse phase, any malformed configuration file should trigger an error condition and use of the regex information should not be allowed. Give some reasonable feedback as to where the parse error occurs.

You will demonstrate that your regex is working by creating some test configuration files and printing out relevant configuration information.

NOTE: I am leaving this fairly open ended on purpose. Any assumptions you make that are beyond the specification of the configuration file should be documented. You should not change the basic functionality of the config file.

NOTE: Be careful of gotchas! E.g. what if I put a # inside a quote? What about quotes in quotes? Do reasonable testing to make sure your regex is handling malformed cases appropriately.

Example File:

```
# Sample config file
# For my new wonderful tower defense game!

[globals]
player=”chris”
chartype=”blue tower”
attributes={10; 20; 40; 70}
multilist={“bob”, 10, true}
[video]
resolution=”1080x700”
scalefactor=1.0f
multidisplay=true

[video:acceleration]
DX12Flag=true
```

```
[audio]
surround=
headphones=whatever # This is a malformed line
@=trueifIfeellikeit # Really should break!

[] # This should break!
```