# threads (condition variables), and formatting via stream manipulators

ESE 2025

# std::condition_variable

The condition_variable class is a synchronization primitive that can be used to block a thread, or multiple threads at the same time, until another thread both modifies a shared variable (the condition), and notifies the condition_variable.

The thread that intends to modify the variable has to

1. acquire a std::mutex (typically via std::lock_guard)
2. perform the modification while the lock is held
3. execute notify_one or notify_all on the std::condition_variable (the lock does not need to be held for notification)

*please note, the above and the following material on condition variables is taken from: https://en.cppreference.com/w/cpp/thread/condition_variable

# std::condition_variable (cont'd)

Any thread that intends to wait on std::condition_variable has to

A. acquire a std::unique_lock<std::mutex>, on the same mutex as used to protect the shared variable either
   a. check the condition, in case it was already updated and notified
   b. execute wait, wait_for, or wait_until. The wait operations atomically release the mutex and suspend the execution of the thread. When the condition variable is notified, a timeout expires, or a spurious wakeup occurs, the thread is awakened, and the mutex is atomically reacquired. The thread should then check the condition and resume waiting if the wake up was spurious.

   or

B. use the predicated overload of wait, wait_for, and wait_until, which takes care of the three steps above

# ASIDE: what do we mean by “atomic” in C++?

- atomic types allow multiple threads to use them without “data races”

- in other words, if a thread writes to an atomic object while another thread reads from the same atomic object AT THE SAME TIME, the read/write operations are defined in order to make this process consistent and predictable each time;

# C++ streams

- streams are sequences of characters, intended for I/O manipulation;
- C++ provides three basic streams, available via three separate headers, namely:
  - iostream: console-oriented streams
  - fstream: streams for reading and writing to files
  - sstream: streams for reading/writing to strings
- it’s useful to note that iostream, fstream and stringstream classes are derived from the more general stream classes of ostream (output streams) and istream (input streams)

# C++ stream manipulators

- just as C has format specifiers, C++ provides a number of stream manipulators that can modify how streams are arranged
- you are already familiar with one manipulator: **std::endl**, starts a new line (you can also start a new line by using the “\n” escape-character sequence, but although the effect is the same, the way each of these new-lines are created are quite different!)

# C++ stream manipulators (cont’d)

- controlling output formats: #include<iostream>
- std::boolalpha/std::noboolalpha, std::oct, std::hex, std::dec
- useful when using these: std::showbase

    example:

```
int x=34823;
std::cout << std::showbase << std::hex;
std::cout << “your number is :” << x;

your number is: 0x8807
```

# C++ stream manipulators (cont’d)

- certain manipulators take arguments, and they are found in the iomanip header, so you’ll need #include <iomanip> for those
- controlling floating-point formats: precision, notation (scientific or decimal)
- cout.precision(): returns current precision
- std::setprecision(): specifies the number of significant digits
- std::scientific, std::fixed;
- printing columns: try std::setw;
    - cout << “i: “ << setw(12) << i; // pads spaces to left of number to give 12 total spaces
- also: setfill(ch), setbase(b) ----> look up!
- there are LOTS of manipulators to explore!

# stream random access

- rather specialized, platform dependent
- std::cout, std::cin DO NOT permit random access
- random access means that we can reposition the stream arbitrarily
  - for example, random access allows us to read the last line, then the first line, then any line in between etc.
- *seek* and *tell* functions are required, and are available for file and string streams (fstream and sstream)