CSE 333 Reference Sheet (Final)

C Library Header – stdlib.h

```
EXIT_SUCCESS  // success termination code
EXIT_FAILURE  // failure termination code

void   exit (int status);                 // terminate calling process
```

Error Library – errno.h

```
errno         // # of the last error, usually checked against defined consts

EAGAIN        // try again
EBADF         // bad file/directory/socket descriptor
EINTR         // interrupted function
EWOULDBLOCK   // operation would block
```

C++ Standard Template Library – vector, list, map, etc.

```
.begin()     // get iterator to beginning (first element)
.end()       // get iterator to end (one past last element)
.size()      // get container size
.erase(...)  // erase element (pass 1 iterator) or range (pass 2 iterators)

template <class T> class std::vector;
   •  .operator[](), .push_back(), .pop_back()
template <class T> class std::list;
   •  .push_back(), .pop_back(), .push_front(), .pop_front(), .sort()
template <class Key, class T> class std::map;
   •  .operator[](), .insert(), .find(), .count()
template <class T1, class T2> struct std::pair
   •  .first, .second
```

C++ STL Algorithms – algorithm

```
std::find()          // returns iterator to element in range that matches val
std::for_each()      // apply function to each element in range
std::min_element()   // get iterator to smallest element in range
std::max_element()   // get iterator to largest element in range
std::sort()          // sorts range into ascending order
```

C++ Smart Pointers Library – memory

```
template <class T> class unique_ptr;
   •  .get(), .reset(), .release()
template <class T> class shared_ptr;
   •  .get(), .use_count(), .unique()
template <class T> class weak_ptr;
   •  .lock(), .use_count(), .expired()
```

POSIX Headers – unistd.h, arpa/inet.h, netdb.h

```
ssize_t read (int fd, void* buf, size_t count);
ssize_t write (int fd, const void* buf, size_t count);
int getaddrinfo (const char* hostname, const char* service,
                 const struct addrinfo* hints, struct addrinfo** res);
int socket (int domain, int type, int protocol);
int connect (int fd, const struct sockaddr* addr, socklen_t addrlen);
int bind (int sockfd, const struct sockaddr* addr, socklen_t addrlen);
int listen (int sockfd, int backlog);
int accept (int sockfd, struct sockaddr* addr, socklen_t* addrlen);
```

Pthreads Header – pthread.h

```
pthread_t            // data type to identify a thread
pthread_mutex_t      // data type for a mutex

int pthread_create (pthread_t* thread, const pthread_attr_t* attr,
                    void* (*start_routine)(void*), void* arg);
int pthread_join (pthread_t thread, void** retval);
int pthread_detach (pthread_t thread);

int pthread_mutex_init (pthread_mutex_t* mutex,
                        const pthread_mutexattr_t* attr);
int pthread_mutex_lock (pthread_mutex_t* mutex);
int pthread_mutex_unlock (pthread_mutex_t* mutex);
int pthread_mutex_destroy (pthread_mutex_t* mutex);
```