2/11/2020 S. Week 6 Tues

xvector.h:
- allocator<Type> alloc; // within class xvector_base, from std library
  - takes zero bytes ONLY IF xvector_base has other non-zero byte-sized members
  - struct foo {}; sizeof => 1
  - struct bar { int a; foo b; }; sizeof => 4
- // begin() can be defined in two-ways: const & non-const
  // cbegin(), however, cannot be two. It has to return const
  - for (auto & i : c) } both use begin(), albeit two diff. types
  - for (const auto & i : c) }
- xvector_iterator: for colon for loop: iterator requires:
  1. *i (operator *), 2. ++i (operator++), 3. i != j; (operator!=)

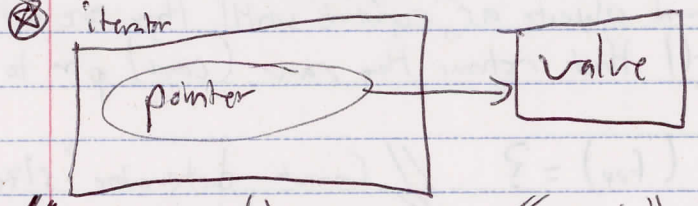
Input Output

Forward

Bidirectional

Rand Access

ALL algorithms MUST assume forward iterators.
↳ Do not assume bidir. or rand. access!



- operator->() returns pointer
- operator*() returns &value

// operator++(): prefix: "++i"; • operator++(int): "i++"
// operator==() and operator!=() can use one another, need to just define one of them.
// Implicit conversion of iterator to const_iterator:
operator xvector_iterator<const value_type>() const // Use for functions needing const iterators
{ return xvector_iterator<const value_type>(*this); }.

```
T operator=(const T& that) {
    if (this != &that) { // VERY important check!
        //...
    }
    return *this;
}
```

Cont...

cont...

xvector.tec:

// std::uninitialized_copy(...): needed since we are copying into something that's uninitialized

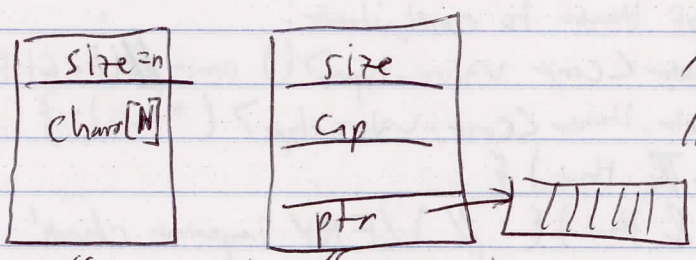
- // std::move(...): not a real function! Matter a copy, turn into a move context
// xvector(initializer_list<value-type> list): Given a list of parms to a ctor
// xvector::reserve(): reserves space in vector to prevent heap churn when we pre-know what vector size will be (at least)
// base.alloc_construct(base, end++, that): Doesn't allocate; only constructs:
new: ① Allocates (malloc()) ② Calls object's ctor
delete: ① Calls object's dtor ② Deallocates (free())
push_back(const value-type& that) vs. push_back(value-type&& that)
⊗ Use the latter when passing in a temp object
- pop_back(): Should it throw an exception? Not necessarily...
↳ at()? Yes!!!
- operator==(xvector& lhs, xvector& rhs)
// Look in xvector.tec, {this is a very important function!!!}
- post-fix operator++: doesn't need to be a member function
- // Note on line 50: for copy ctor, you have to dealloc vector that you're putting over

String Representation: using string = basic_string<char>; string != vector<char>
↳ A string is similar to vectors, but they are NOT the same!
⊗ Copy-on-write: Pass around objects as if they were constant until they are written to, then copy
↳ They should have c_str() that returns the raw (const) ptr to underlying string

struct Foo {
  int n;
  char s[n];
};

sizeof(Foo) = ? // Cannot determine size of foo // at compile time, but required to...

Other Impl: // Union of the two, use one or the other // depn. on size.



Cont...

```
struct foo {
    int n;
    char B;
} x, y, z;   <- why we need semicolon, could produce var's x, y, z
```

This should be "class", really...

(Cont...)

```
struct opt-string {     // Works only on a little-endian machine
    struct long_string {        ↑ Why is this?
        size_t size_;
        size_t capacity;
        char* data_;
    };
    struct short-string {
        unsigned char size_ {0};
        char data_ [sizeof(long_string) - 1] {};
    };
    union {                     // Not portable, but can exchange optimization
        long_string long_;      // for portability
        short_string short_;
    }
    bool is_long() const {
        return short_.size_ & 1;
    }
    // ...
};
```

int x = 0xffefcdab

Little-End: &x+3 ... &x

0xff	0xef	0xcd	0xab

"Big part last"

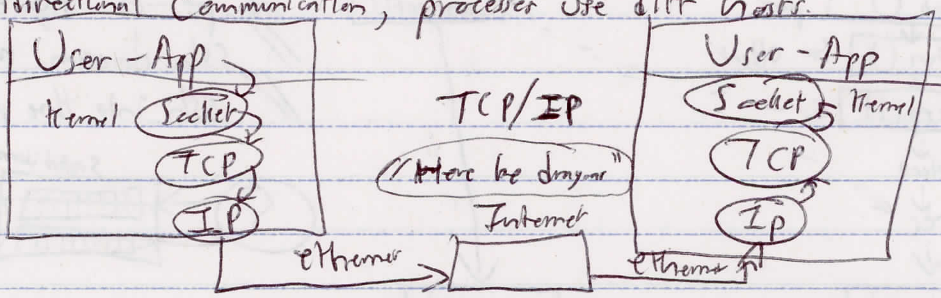
Big-Endian: &x ... &x+3

0xff	0xef	0xcd	0xab

"Big part first"

- sizeof(std::vector) is 24 bytes: 3 ptr
  - ↳ But, sizeof(std::string) is 16 bytes: 1 ptr, 1 capacity

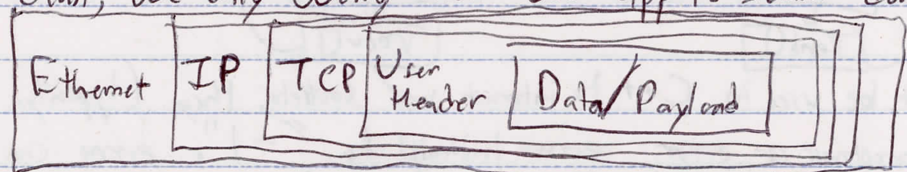
Sockets: Bidirectional communication, processes use diff hosts.



⊗ Sockets are treated like "files": you read & write to them

⊗ This class, we only worry about User-app to Socket domain.

Packets:



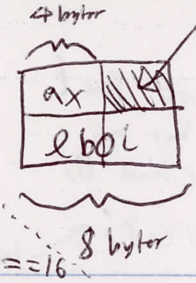
// Daemon: process that sleeps on a computer in the background til woken up

(Cont...) ↓

Page 4! (Cont...)

```
struct foo {
    int ax;
    long lbol;
};
```

sizeof(foo) == 16



"Cookies spy on you"
↳ No really. They do...

Socket Types:

	TCP stream	UDP datagram
reliable	yes	no
msg boundary	no	yes
Connection	yes	no

- Ports: Lower # are reserved (up til ~1023): You CANNOT use reserved ports!!!
  ↳ 0 to 65535
- 32-bit IP (v4): 4 octets: 1 IP per about 2 people on this planet
  - 128-bit ↳ v6: 32 octets: Looooots of addresses, but not *fully* standard yet...
  - ↳ We still use the dumb v4 version :(

	(u)int8_t	should be (unsigned) char?
Port #	(u)int16_t	" " (unsigned) short?
IPv4	(u)int32_t	" " (unsigned) int?
	(u)int64_t	" " (unsigned) long? // long ~ sizeof(ptr)

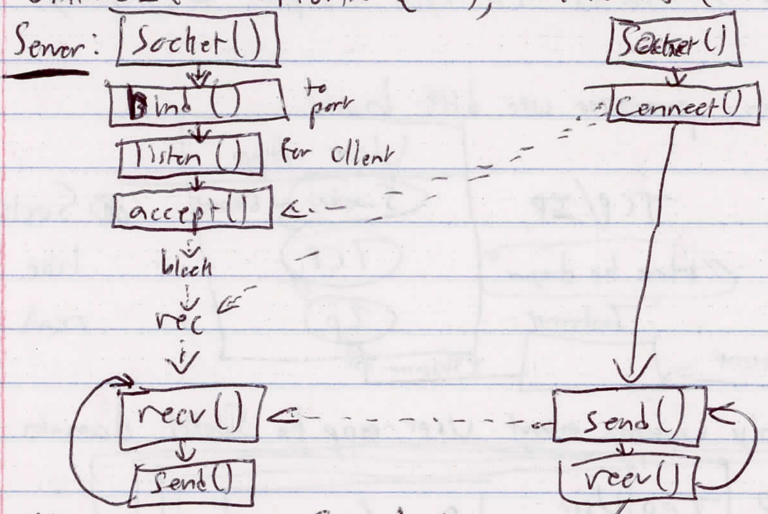
- ★ hostname: has to be looked up to determine IP addr (see: CSE 150)

- Big Endian: Motorola 68000, Sun Sparc, Network
- Little Endian: DEC alpha, Intel x86, AMD

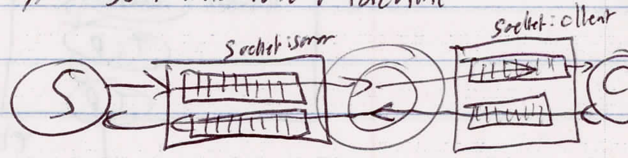
0x2c89436a ⇕ 0x6a43892c

- uint16_t: htons(...), ntohs(...)
  uint32_t: htonl(...), ntohs(...)

Sockets:



// ps -ef | grep sshd
ssh daemon, must run in order to ssh into the machine



System Calls: Must be used in C++ to interact w/ sockets; they (typically) return a result to check for correctness or error: errors indicated by "-1". errno can be checked, but errno isn't thread safe! Use "strerror(errno);" to get English descr of error.

- Sockaddr and sockaddr_in: Web Search to see internals of struct

END