STL iterator

- Each container class has an associated iterator class (e.g. `vector<int>::iterator`) used to iterate through elements of the container
  - Iterator range is from `begin` up to `end` i.e., `[begin, end)`
  - `end` is one past the last container element!
- Some container iterators support more operations than others
  - All can be incremented (`++`), copied, copy-constructed
  - Some can be dereferenced on RHS (e.g. `x = *it;`)
  - Some can be dereferenced on LHS (e.g. `*it = x;`)
  - Some can be decremented (`--`)
  - Some support random access (`[[]`, `+=`, `-=` operators)

Range for Statement (C++11)

- Syntactic sugar similar to Java's `foreach`
  ```cpp
  for ( declaration : expression ) { statements }
  ```
  - `declaration` defines loop variable
  - `expression` is an object representing a sequence
    - Strings, initializer lists, arrays with an explicit length defined, STL containers that support iterators
  ```cpp
  // Prints out a string, one character per line
  std::string str("hello");
  for ( auto c : str ) {
    std::cout << c << std::endl;
  }
  ```

STL list

- A generic doubly-linked list
  - Elements are not stored in contiguous memory locations
    - Does not support random access (e.g. cannot do `list[5]`)
  - Some operations are much more efficient than vectors
    - Constant time insertion, deletion anywhere in list
    - Can iterate forward or backwards
    - Has a built-in sort member function
  - Doesn’t copy! Manipulates list structure instead of element values

STL map

- One of C++'s associative containers: a key/value table, implemented as a search tree
  - General form: `map<key_type, value_type> name;`
  - Keys must be `unique`
    - `multimap` allows duplicate keys
  - Efficient lookup (O(log n)) and insertion (O(log n))
    - Access value via `name[key]`
      - if key doesn’t in map, it is added to the map
  - Elements are type `pair<key_type, value_type>` and are stored in sorted order (key is field `first`, value is field `second`)
    - Key type must support `less-than` operator `(<)`
Poll Everywhere

Should we use a reference?

A. We must NOT use a reference
B. It’s OK but discouraged to use a reference
C. It’s OK and encouraged to use a reference
D. We must use a reference
E. We’re lost...

- Provided are three different ways to read the contents of a file. Rank the implementations by their efficiency.
  - Assume that buffers are allocated and files opened/closed for you

```c
fread(buf, LEN, sizeof(char), file);
return buf;
```

Implementation #1

```c
while(read(fd, buf+numread, sizeof(char)) != 0) {
    numread += sizeof(char);
    return buf;
}
```

Implementation #2

```c
while((res = read(fd, buf+numread, LEN - numread)) != 0) {
    numread += sizeof(char);
}
```

Implementation #3

What is wrong with this program? (ignoring style issues)

```c
#define FOO 333
struct pair {
    int x, y;
};
```

#include "pair.h"
#include "stdio.h"

void Pair_Allocate(pair *out) {
    out = (pair *) malloc(sizeof(pair));
    out->x = 0;
    out->y = 0;
}

void Pair_Print(pair *p) {
    printf("(x:%d, y:%d)\n", p->x, p->y);
}

main.c

```c
main() {
...
}
```

util.h

```c
#include "pair.h"
#include "util.h"
```