

STL iterator
Specific to container and element type

- Each container class has an associated **iterator** class (e.g. `vector<int>::iterator`) used to iterate through elements of the container
 - <http://www.cplusplus.com/reference/std/iterator/>
 - 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)

5

5

Range for Statement (C++11)

- Syntactic sugar similar to Java's `foreach`

```
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

`str = sequence of characters`

```
// Prints out a string, one
// character per line
std::string str("hello");

for ( auto c : str ) {
    std::cout << c << std::endl;
}
```

9

9

STL list

- A generic doubly-linked list
 - <http://www.cplusplus.com/reference/stl/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 *Iterate backward: --* *Iterate forward: ++*
- Has a built-in sort member function
 - Doesn't copy! Manipulates list structure instead of element values

13

13

STL map

- One of C++'s *associative* containers: a key/value table, implemented as a search tree
- <http://www.cplusplus.com/reference/stl/map/>
- 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 exist, 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 (`<`)

15

15

W UNIVERSITY of WASHINGTON L14: C++ Standard Template Library CSE333, Summer 2020

Poll Everywhere pollev.com/cse33320su

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...

```
#ifndef _COMPLEX_H_
#define _COMPLEX_H_

#include <iostream>

namespace complex {
    class Complex {
        public:
            // Copy constructor, should we
            // pass a reference or not? (Answer: ?)
            Complex(const Complex &copyme) {
                real_ = copyme.real_;
                imag_ = copyme.image_;
            }

        private:
            double real_, imag_;
    }; // class Complex
} // namespace complex
#endif // _COMPLEX_H_
```

19

19

W UNIVERSITY of WASHINGTON L14: C++ Standard Template Library CSE333, Summer 2020

Poll Everywhere pollev.com/cse33320su

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...

```
#include <iostream>
namespace complex {

    class Complex {
        public:
            // Should operator+ return a reference or not?
            // (Answer: ?)
            Complex &operator+(const Complex &a) const {
                Complex tmp(0,0);
                tmp.real_ = this->real_ + a.real_;
                tmp.imag_ = this->imag_ + a.imag_;
                return tmp;
            }

        private:
            double real_, imag_;
    }; // class Complex
} // namespace complex
```

21

21

W UNIVERSITY of WASHINGTON L14: C++ Standard Template Library CSE333, Summer 2020

Poll Everywhere pollev.com/cse33320su

- ❖ 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

```
fread(buf, LEN, sizeof(char), file);           Implementation #1
return buf;

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

Implementation #2

```
while((res = read(fd, buf+numread, LEN - numread) != 0) {
    // ...
    numread += res;
}
return buf;
```

Implementation #3

23

23

W UNIVERSITY of WASHINGTON L14: C++ Standard Template Library CSE333, Summer 2020

Poll Everywhere pollev.com/cse33320su

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

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

pair.h

```
#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)", p.x, p.y);
}
```

util.h

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

int main() {
    pair *p;
    Pair_Allocate(p);
    p->x = FOO;
    p->y = 351;
    Pair_Print(*p);
}
```

main.c

25

25