Creative Software Programming

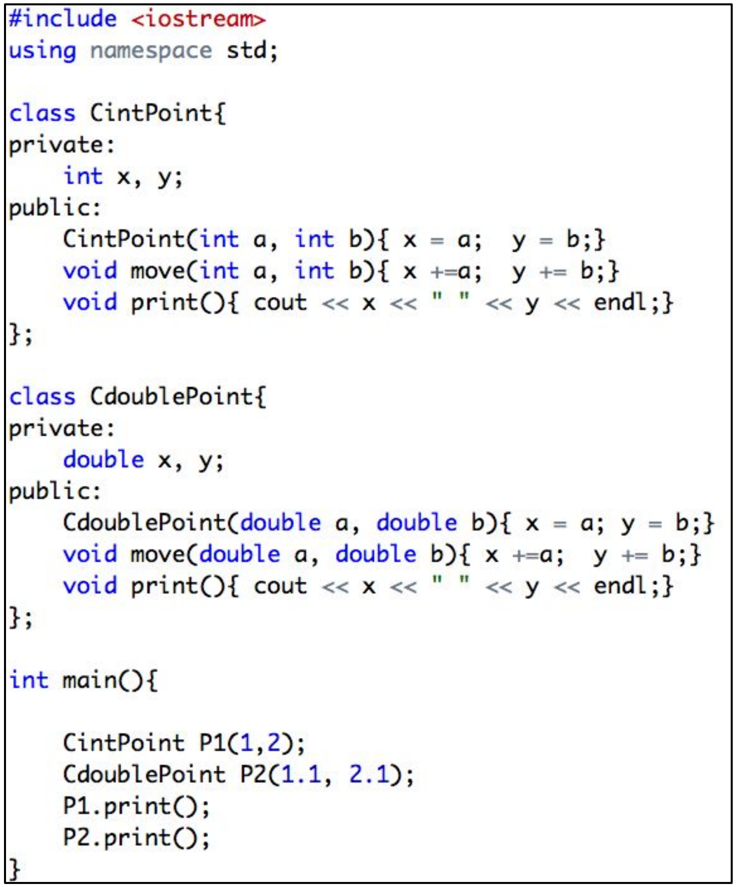
7 – Standard Template Library

Today's Topics

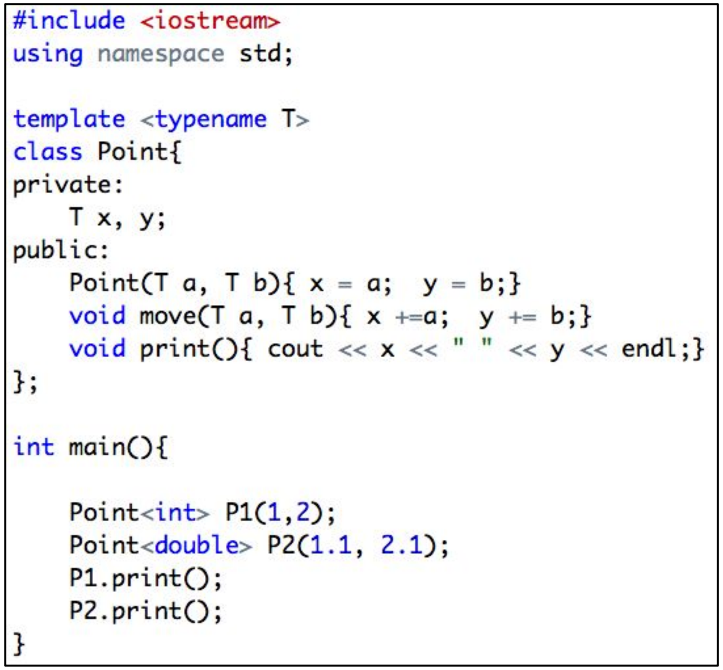
- Intro to Template (briefly)
- STL (Standard Template Library)
- Containters
  - std::vector, std::list
  - std::stack, std::queue
  - std::set, std::map
- Iterator
- std::string

Template

- Templates provide parameterized types.
- Functions and classes can be templated.



```
#include <iostream>
using namespace std;

class CintPoint{
private:
    int x, y;
public:
    CintPoint(int a, int b){ x = a;  y = b;}
    void move(int a, int b){ x +=a;  y += b;}
    void print(){ cout << x << " " << y << endl;}
};

class CdoublePoint{
private:
    double x, y;
public:
    CdoublePoint(double a, double b){ x = a; y = b;}
    void move(double a, double b){ x +=a;  y += b;}
    void print(){ cout << x << " " << y << endl;}
};

int main(){

    CintPoint P1(1,2);
    CdoublePoint P2(1.1, 2.1);
    P1.print();
    P2.print();
}
```



```
#include <iostream>
using namespace std;

template <typename T>
class Point{
private:
    T x, y;
public:
    Point(T a, T b){ x = a;  y = b;}
    void move(T a, T b){ x +=a;  y += b;}
    void print(){ cout << x << " " << y << endl;}
};

int main(){

    Point<int> P1(1,2);
    Point<double> P2(1.1, 2.1);
    P1.print();
    P2.print();
}
```

An example of class template

Standard Template Library (STL)

- STL defines powerful, template-based, reusable components.
- STL uses generic programming based on templates

- A collection of useful template for handling various kinds of data structure and algorithms
  - Containers: data structures that store objects of any type
  - Iterators: used to manipulate container elements
  - Algorithms: operations on containers for searching, sorting and many others

Containers

- Sequence
  - Elements are accessed by their position in the sequence.
  - **vectors**: fast insertion at end, random access
  - **list**: fast insertion anywhere, sequential access
  - **deque** (double-ended queue): fast insertion at either end, random access

- Container adapter
  - “Adapting” the interface of underlying container to provide the desired behavior.
  - **stack**: Last In First Out
  - **queue**: First In First Out

Containers

- Associative container
  - Elements are referenced by their key and not by their absolute position in the container, and maintained in sorted key order.
  - **set**: add or delete elements, query for membership…
  - **map**: a mapping from one type (key) to another type (value)
  - **multimaps**: maps that associate a key with several values

std::vector - a resizable array

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> int_vec;
  for (int i = 0, val; i < 3; ++i) {
    cout << "input: ";
    cin >> val;
    int_vec.push_back(val);
  }
  int org_size = int_vec.size();
  int_vec.resize(org_size + 3);
  for (int i = org_size; i < int_vec.size(); ++i) {
    int_vec[i] = i;
  }
  for (int i = 0; i < int_vec.size(); ++i) {
    cout << int_vec[i];
  }
  cout << endl;
  return 0;
}
```

std::vector - a resizable array

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> vec(3, 0);
  vec.push_back(10);
  vec.push_back(20);
  if (vec.empty() == true) {
    cout << "vec is empty" << endl;
  } else {
    cout << "vec.front: " << vec.front() << endl;
    cout << "vec.back: " << vec.back() << ", size=" << vec.size() << endl;
    vec.pop_back();
    cout << "vec.back: " << vec.back() << ", size=" << vec.size() << endl;
  }
  vec.clear();
  return 0;
}
```

std::vector - a resizable array

- You can make a vector of strings or other classes.

```
#include <string>
#include <vector>
using namespace std;

struct Complex { double real, imag; /* ... */ };

// ...
vector<string> vs;
for (int i = 0; i < 10; ++i) cin >> vs[i];
// vector(size, initial_value)
vector<string> vs2(5, "hello world");

vector<Complex> v1(10);
vector<Complex> v2(10, Complex(1.0, 0.0));
Complex c(0.0, 0.0);
v2.push_back(c);
for (int i = 0; i < v2.size(); ++i) {
  cout << v2[i].real << "+" << v2[i].imag << "i" << endl;
}
```

std::vector - a resizable array

- Sometimes you may want to use a vector of pointers.

```
#include <vector>
using namespace std;

class Student;

vector<Student*> vp(10, NULL);
for (int i = 0; i < vp.size(); ++i) {
  vp[i] = new Student;
}

// After using vp, all elements need to be deleted.

for (int i = 0; i < vp.size(); ++i) delete vp[i];
vp.clear();
```

std::vector

- Element are stored in contiguous storage.
- Random access: Fast access to any element
- Fast addition/removal of elements at the **end** of the sequence.

References for STL

- std::vector
  - http://www.cplusplus.com/reference/vector/vector/
- STL containers
  - http://www.cplusplus.com/reference/stl/
- You can find documents for any other STL features in the links in the above pages.

Iterator

- Iterator: a pointer-like object **pointing to** an element in the container.

- Iterators provide **a generalized way** to traverse and access elements stored in a container.
  - can be ++ or -- (move to next or prev element)
  - dereferenced with *
  - compared against another iterator with == or !=

- Iterators are generated by STL container member functions, such as begin() and end().

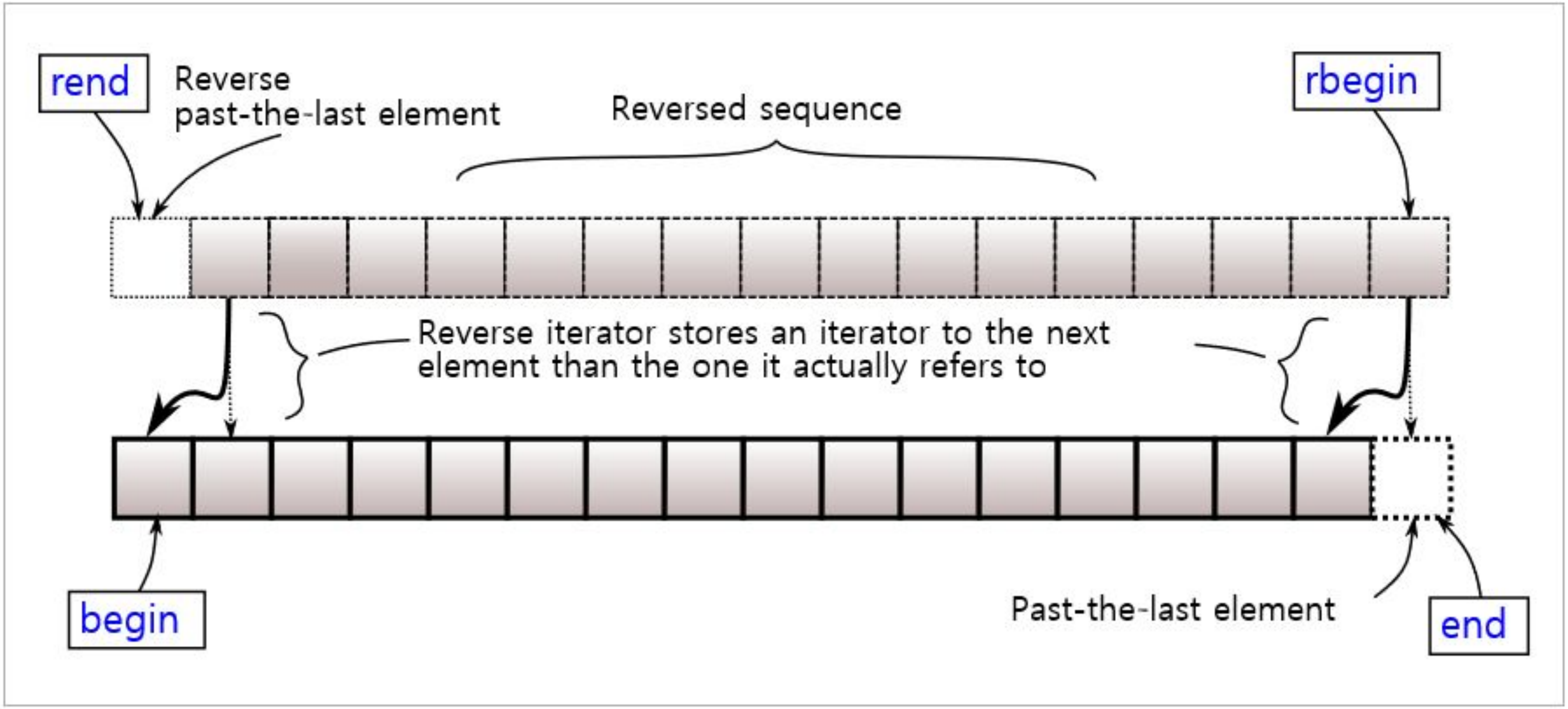
std::vector with iterator

```
#include <iostream>
#include <vector>
using namespace std;

void PrintVec(const vector<int>& vec, const string& name) {
  cout << name;
  for (vector<int>::const_iterator it = vec.begin(); it != vec.end(); ++it) {
    cout << " " << *it;
  }
  cout << endl;
}

int main() {
  vector<int> vec(5);
  vector<int>::iterator it = vec.begin();
  for (int i = 0; i < vec.size(); ++i, ++it) {
    *it = i;
  }
  PrintVec(vec, "vec");
  vec.insert(vec.begin() + 2, 100);
  PrintVec(vec, "vec");
  vec.erase(vec.begin() + 2);
  PrintVec(vec, "vec");
  return 0;
}
```

std::vector with iterator

```
#include <vector>
#include <iostream>
using namespace std;

int main(void) {
  // vector(sz)
  vector<int> v(10);
  for (int i = 0; i < v.size(); ++i) v[i] = i;

  // begin(), end()
  for (vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
  cout << " " << *it;
  }
  // Output:  0 1 2 3 4 5 6 7 8 9

  // rbegin(), rend()
  for (vector<int>::reverse_iterator it = v.rbegin(); it != v.rend(); ++it) {
    cout << " " << *it;
  }
  // Output:  9 8 7 6 5 4 3 2 1 0
  return 0;
}
```

Meaning of begin(), end(), rbegin(), rend()


rend
Reverse
past-the-last element
Reversed sequence
rbegin
Reverse iterator stores an iterator to the next
element than the one it actually refers to
begin
Past-the-last element
end

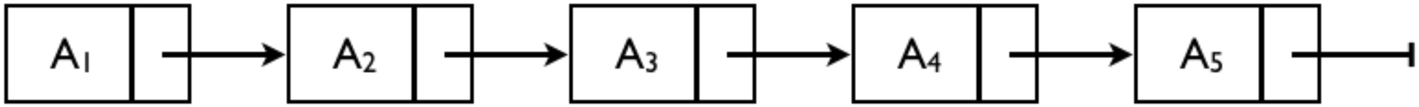
Quiz #1

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> vec(5);
  for (int i = 0; i < vec.size(); ++i) {
    vec[i] = 2 * i;
  }
  vector<int>::iterator it = vec.begin();
  *it = 8;
  *(it + 2) = 9;

  for (int i = 0; i < vec.size(); ++i) {
    cout << vec[i] << " ";
  }
  cout << endl;
  return 0;
}
```

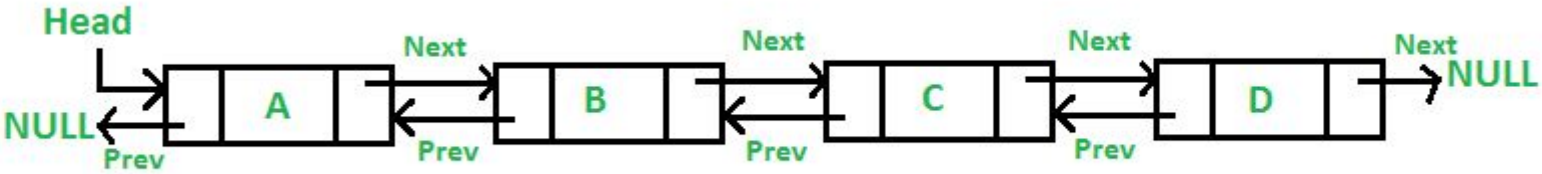
- What is the expected output of this program? (If a compile error is expected, just write down "error").

Concept of Linked List

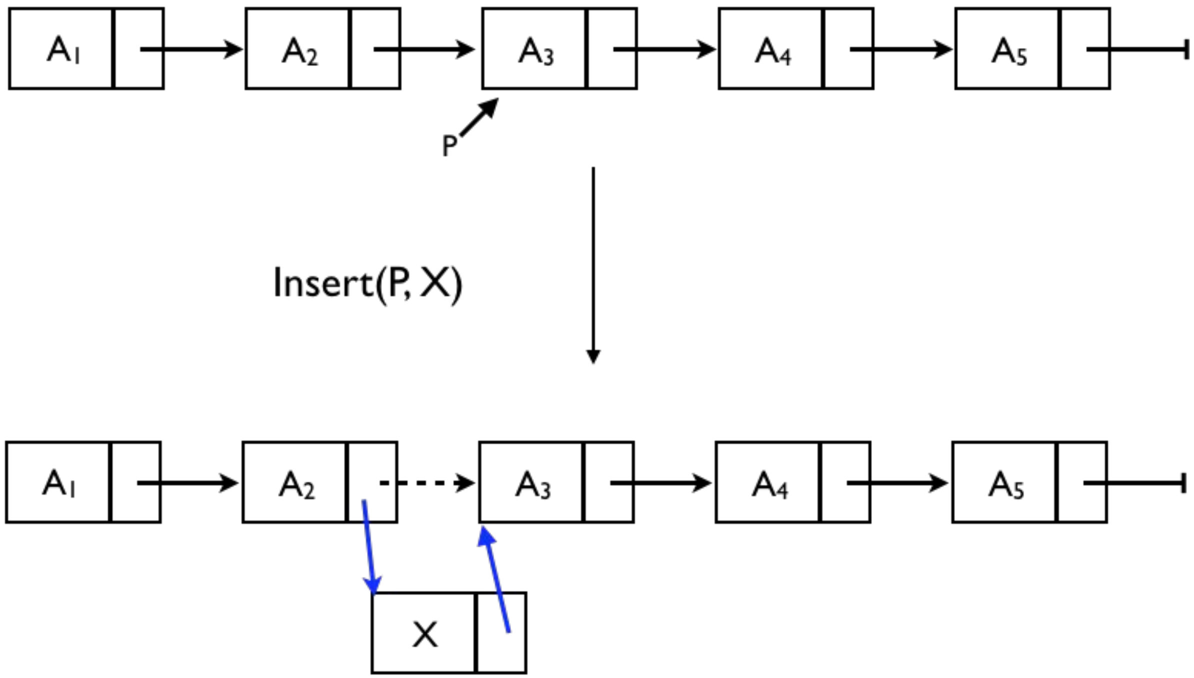
- Singly linked list: A node consists of the data and a link to the next node.

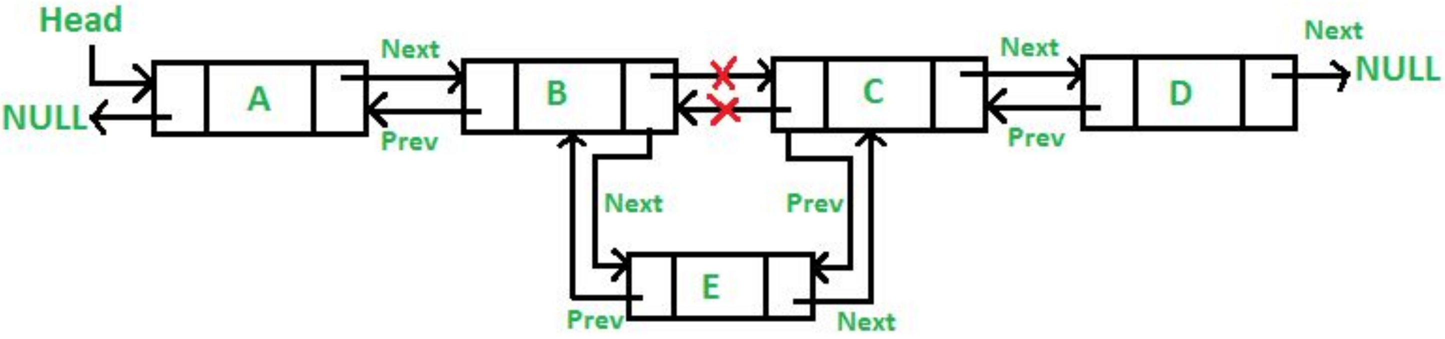


- Doubly linked list: with links to prev. & next node.

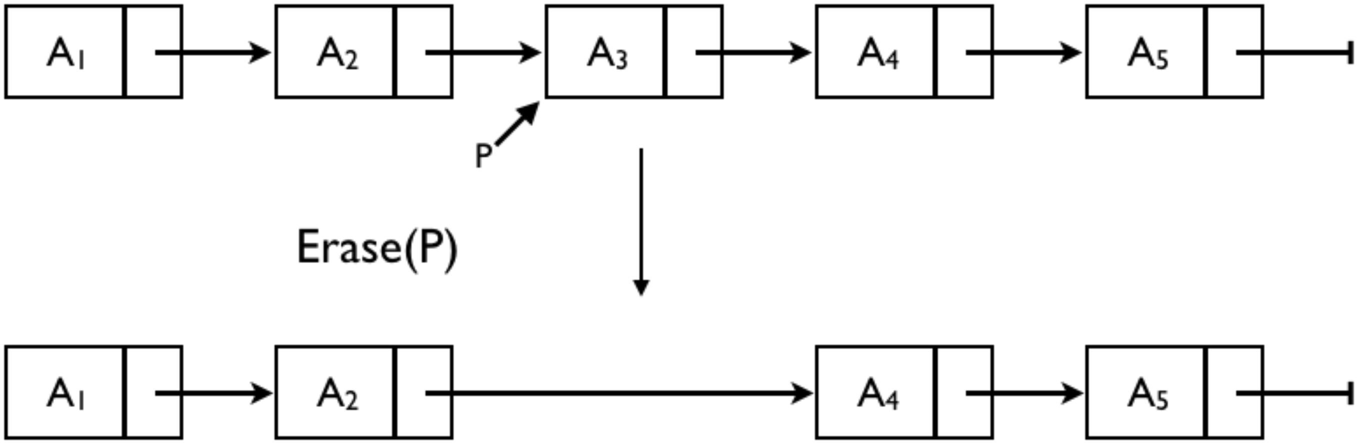


Concept of Linked List: insert





Concept of Linked List: erase



std::list

- Implemented as a doubly-linked list.
  - Non-contiguous storage.

- Sequential access
  - One should iterate from a known position (like begin() or end()) to access to some element.

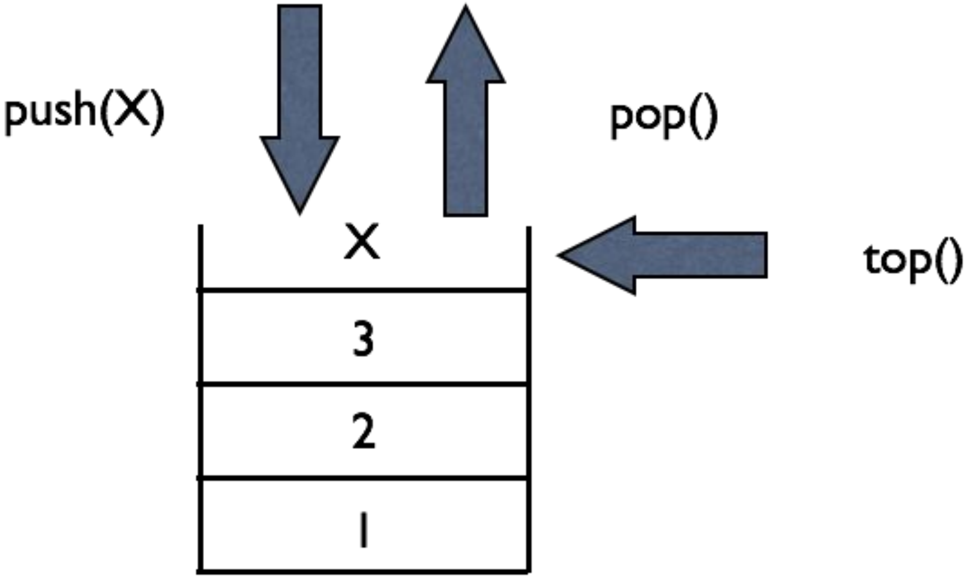
- Fast addition/removal of elements **anywhere** of the sequence.

std::list – an insert and erase example

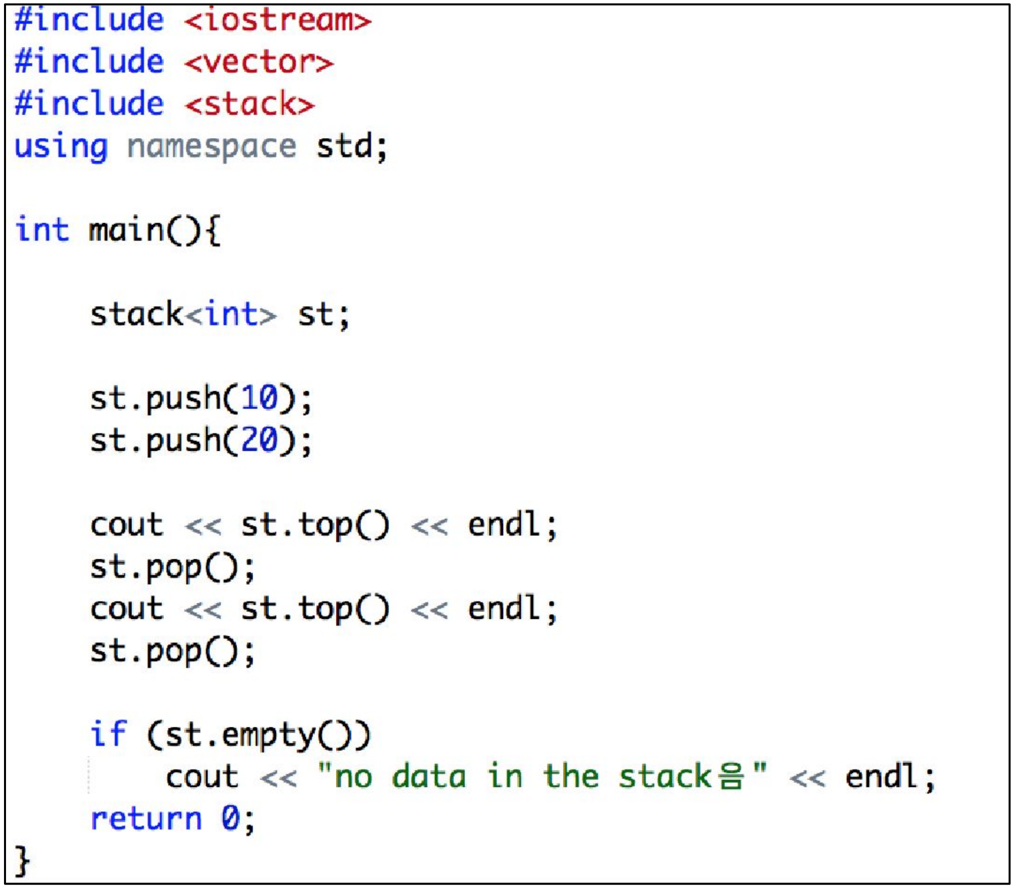
```
#include <iostream>
#include <list>
using namespace std;

void PrintList(const list<int>& lst) {
  for (list<int>::const_iterator it = lst.begin(); it != lst.end(); ++it) {
    cout << " " << *it;
  }
  cout << endl;
}

int main() {
  list<int> lst(5);
  for (int i = 0; i < 5; ++i) {
    lst.insert(lst.end(), i);
  }
  PrintList(lst);           // 0 1 2 3 4
  list<int>::iterator it = lst.begin();
  ++it;
  it = lst.insert(it, 100);
  PrintList(lst);           // 0 100 1 2 3 4

  ++it;
  ++it;
  lst.erase(it);
  PrintList(lst);           // 0 100 1 3 4

  return 0;
}
```

An iterator that points to the first of the newly inserted elements.

Concept of Stack : Last In First Out

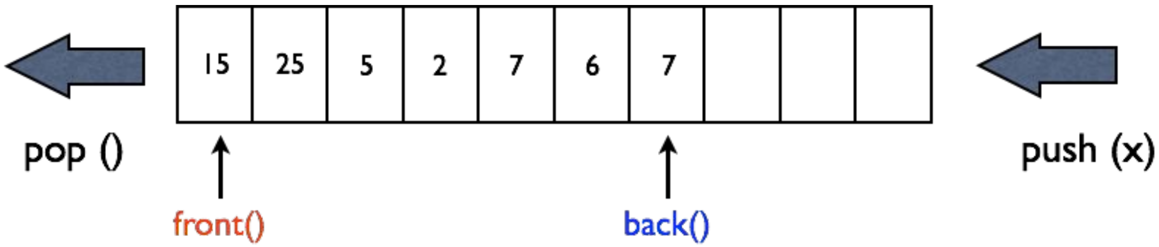


std::stack - example



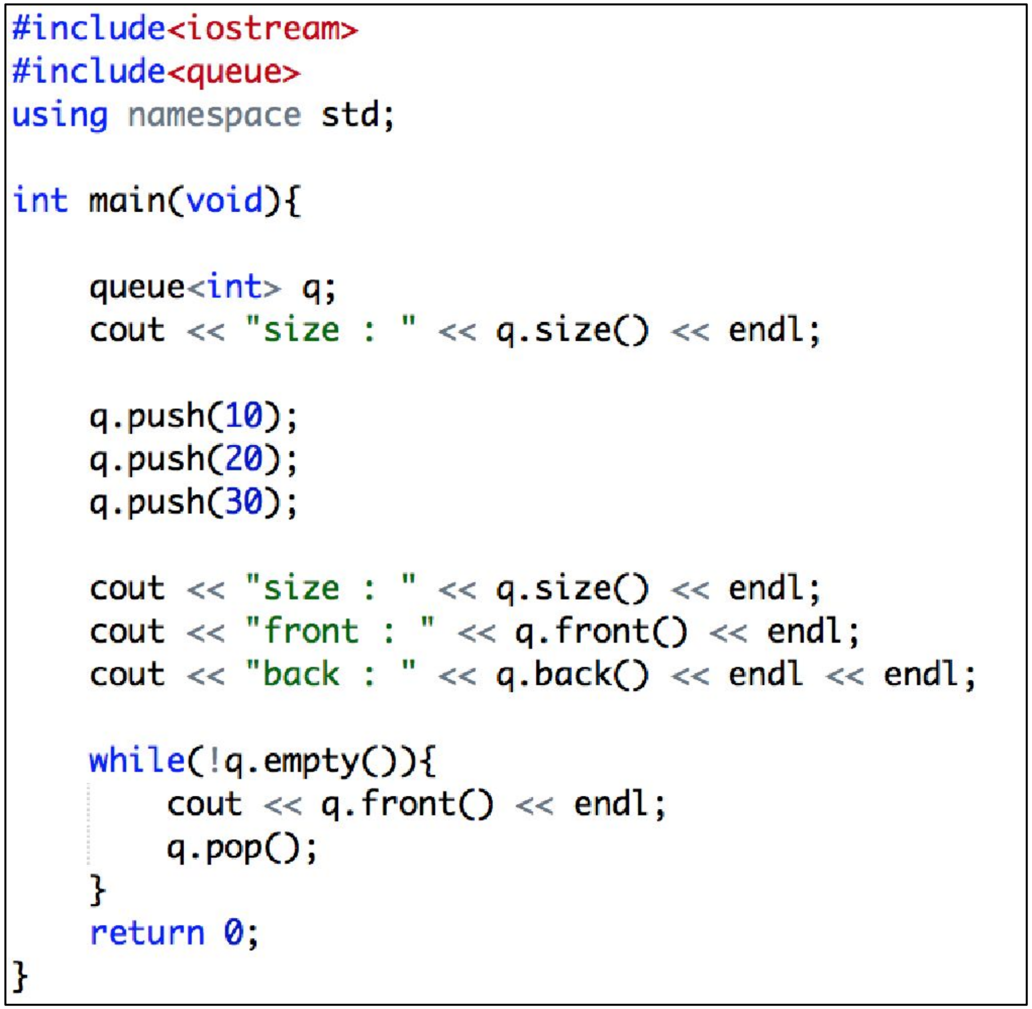
```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

int main(){

    stack<int> st;

    st.push(10);
    st.push(20);

    cout << st.top() << endl;
    st.pop();
    cout << st.top() << endl;
    st.pop();

    if (st.empty())
        cout << "no data in the stack음" << endl;
    return 0;
}
```

Concept of Queue : First In First Out

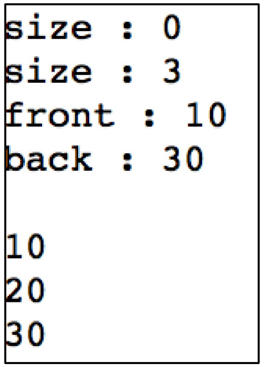


15	25	5	2	7	6	7			

std::queue - example



```
#include<iostream>
#include<queue>
using namespace std;

int main(void){

    queue<int> q;
    cout << "size : " << q.size() << endl;

    q.push(10);
    q.push(20);
    q.push(30);

    cout << "size : " << q.size() << endl;
    cout << "front : " << q.front() << endl;
    cout << "back : " << q.back() << endl << endl;

    while(!q.empty()){
        cout << q.front() << endl;
        q.pop();
    }
    return 0;
}
```



```
size : 0
size : 3
front : 10
back : 30

10
20
30
```

Other Vector-like Containers

- List, stack, queue, and deque (double-ended queue).

	vector	**list**	**stack**	**queue**	**deque**
Random access	operator[] at()	-	-	-	operator[] at()

Sequential access	front()
	back()
		front()
		back()
			top()	front()
				back()
					front()
					back()

Iterators	begin(), end()
	rbegin(), rend()
		begin(), end()
		rbegin(), rend()
			-	-	begin(), end()
					rbegin(), rend()

Adding elements	push_back()
	insert()
		push_front()
		push_back()
		insert()
			push()	push()	push_front()
					push_back()
					insert()

Deleting elements	pop_back()
	erase()
	clear()
		pop_front()
		pop_back()
		erase()
		clear()
			pop()	pop()	pop_front()
					pop_back()
					erase()
					clear()

Adjusting size	resize()
	reserve()
		resize()	-	-	resize()

std::map

- Contains key-value pairs with **unique keys**.

- Associative: Elements are referenced by their key, and maintained in sorted key order.

- Accessing with keys is efficient.

std::map - example

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

void PrintMap(const map<string, double>& m) {
  for (map<string, double>::const_iterator it = m.begin(); it != m.end(); ++it) {
    cout << " (" << it->first << ", " << it->second << ")";;
  }
  cout << endl;
}

int main() {
  map<string, double> m;
  for (int i = 0; i < 4; ++i) {
    m.insert(make_pair("str" + to_string(i), i * 0.5));
  }
  PrintMap(m);    // (str0, 0) (str1, 0.5) (str2, 1) (str3, 1.5)
  m["pi"] = 3.1415;
  PrintMap(m);    // (pi, 3.1415) (str0, 0) (str1, 0.5) (str2, 1) (str3, 1.5)

  map<string, double>::iterator it = m.find("pi");
  if (it == m.end()) {
    cout << "not found" << endl;
  } else {
    cout << "find: " << it->first << " = " << it->second << endl;
  }               // find: pi = 3.1415
  return 0;
}
```

std::set

- Contains **unique keys**.

- Associative: Elements are referenced by their key, and maintained in sorted key order.

- Accessing with keys is efficient.

std::set - example

```
#include <iostream>
#include <set>
using namespace std;

set<int> s;
for (int i = 0; i < 10; ++i) s.insert(i * 10);

for (set<int>::const_iterator it = s.begin(); it != s.end(); ++it) {
  cout << " " << *it;  // s: 0 10 20 30 40 50 60 70 80 90
}
cout << s.size();
cout << s.empty();

set<int>::iterator it, it_low, it_up;
it = s.find(123);  // it == s.end()
it = s.find(50);   // s: 0 10 20 30 40 50 60 70 80 90
                   //                  ^it
s.clear();         // s:
```

Other associative containers

- Multiset and multimap allows duplicate keys.

```
#include <iostream>
#include <set>
#include <map>
using namespace std;

int main() {
  set<int> s;
  map<int, int> m;
  multiset<int> ms;
  multimap<int, int> mm;
  for (int i = 0; i < 10; ++i) {
    int key = i / 2;
    pair<int, int> p(key, i);
    s.insert(key), ms.insert(key);
    m.insert(p), mm.insert(p);;
  }
  cout << "s: " << s.size() << ", ms: " << ms.size();  // s: 5, ms: 10

  for (set<int>::iterator it = s.begin(); it != s.end(); ++it) {
    cout << " " << *it;   // 1 2 3 4 5
  }
  for (multiset<int>::iterator it = ms.begin(); it != ms.end(); ++it) {
    cout << " " << *it;   // 1 1 2 2 3 3 4 4 5 5
  }
  return 0;
}
```

Quiz #2

```
#include <iostream>
#include <map>
using namespace std;

int main() {
  map<string, int> prices;
  prices["orange"] = 10;
  prices["apple"] = 20;
  prices["tomato"] = 15;

  map<string, int>::iterator it;
  it = prices.find("apple");
  cout << it->second << endl;

  return 0;
}
```

- What is the expected output of this program? (If a compile error is expected, just write down "error").

Iterator again

- Iterators provide **a generalized way** to traverse and access elements stored in a container.

- Iterators serve as **an interface** for various kinds of containers.

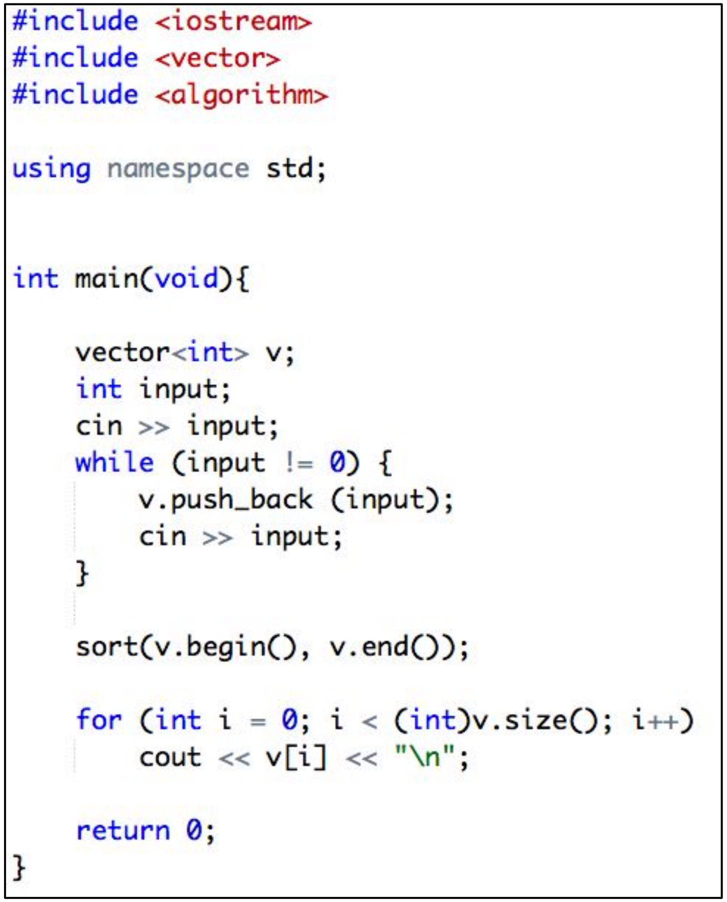
- Passing and returning iterators makes an algorithms more generic, because the algorithms will work for **any** containers.

Algorithm

- Many useful algorithms are available
  - sort
  - min, max, min_element, max_element
  - binary_search

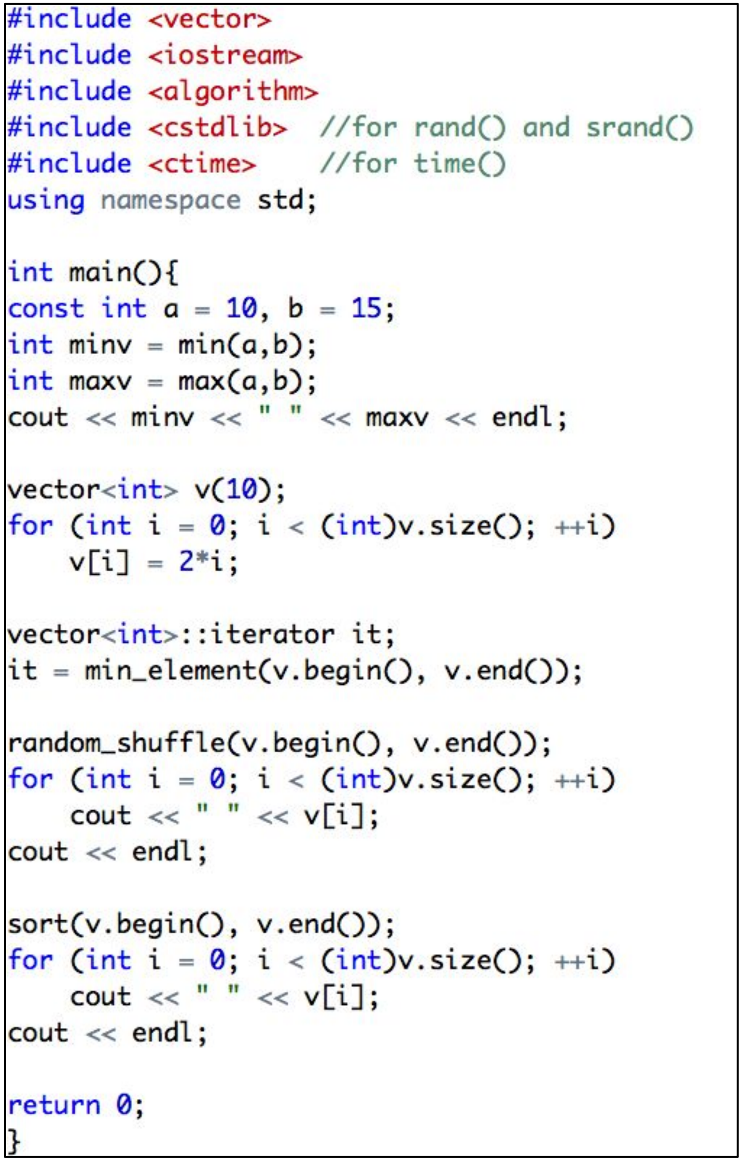
std::sort

void sort(RandomAccessIterator first, RandomAccessIterator last);
Void sort(RandomAccessIterator first, RandomAccessIterator last, Compare comp)



```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;


int main(void){

    vector<int> v;
    int input;
    cin >> input;
    while (input != 0) {
        v.push_back (input);
        cin >> input;
    }

    sort(v.begin(), v.end());

    for (int i = 0; i < (int)v.size(); i++)
        cout << v[i] << "\n";

    return 0;
}
```

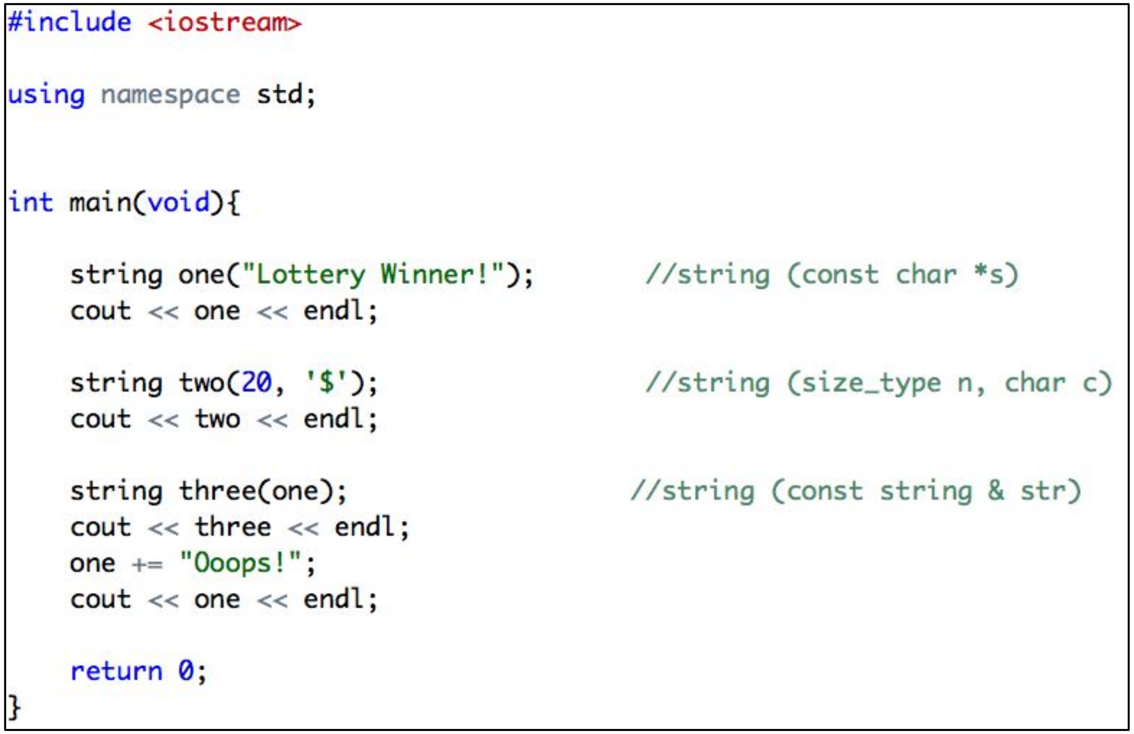
std::min, std::max, std::min_element, std::max_element



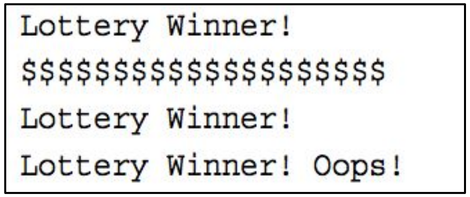
```cpp
#include <vector>
#include <iostream>
#include <algorithm>
#include <cstdlib>  //for rand() and srand()
#include <ctime>    //for time()
using namespace std;

int main(){
const int a = 10, b = 15;
int minv = min(a,b);
int maxv = max(a,b);
cout << minv << " " << maxv << endl;

vector<int> v(10);
for (int i = 0; i < (int)v.size(); ++i)
    v[i] = 2*i;

vector<int>::iterator it;
it = min_element(v.begin(), v.end());

random_shuffle(v.begin(), v.end());
for (int i = 0; i < (int)v.size(); ++i)
    cout << " " << v[i];
cout << endl;

sort(v.begin(), v.end());
for (int i = 0; i < (int)v.size(); ++i)
    cout << " " << v[i];
cout << endl;

return 0;
}
```

std::string - constructor

- In C++, STL provides a powerful string class.



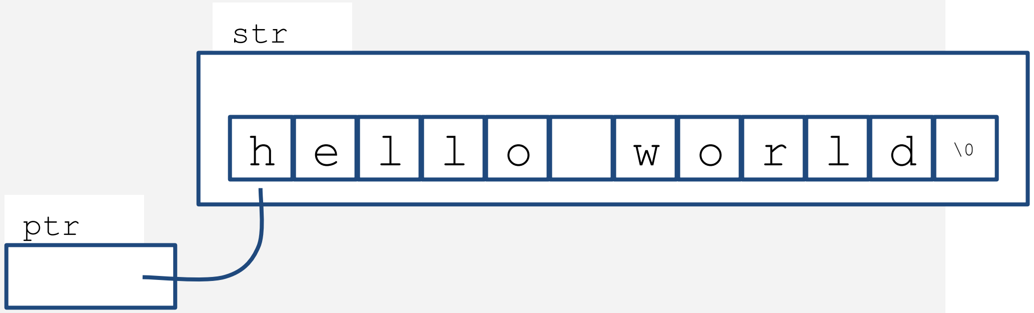
```
#include <iostream>

using namespace std;


int main(void){

    string one("Lottery Winner!");       //string (const char *s)
    cout << one << endl;

    string two(20, '$');                 //string (size_type n, char c)
    cout << two << endl;

    string three(one);                  //string (const string & str)
    cout << three << endl;
    one += "Ooops!";
    cout << one << endl;

    return 0;
}
```



```
Lottery Winner!
$$$$$$$$$$$$$$$$$$$$
Lottery Winner!
Lottery Winner! Oops!
```

(Recall) std::string - c_str()

- Returns a pointer to a null-terminated string array representing the current value of the string object.



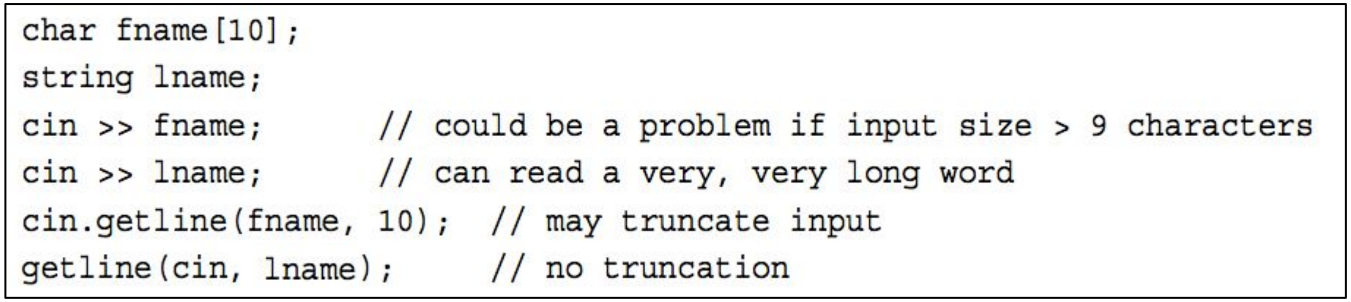
```
#include <string>

std::string str = "hello world";
const char* ptr = str.c_str();
printf("%s\n", ptr);

// ...


std::string str1 = str + " - bye world";
assert(str1 == "hello world - bye world");

assert(str.length() > 10);
assert(str[0] == 'h');
str[0] = 'j';
str.resize(5);
assert(str == "jello");

// check out http://www.cplusplus.com/reference/string/string/
// resize(), substr(), find(), etc.
```

(Recall) std::string - input

```
std::string str;

std::cin >> str; // read a word (separated by a space, tab, enter)

std::getline(cin, str); // read characters until the default
                        // delimiter '\n' is found

std::getline(cin, str, ':'); // read characters until the delimiter
                             // ':' is found
```

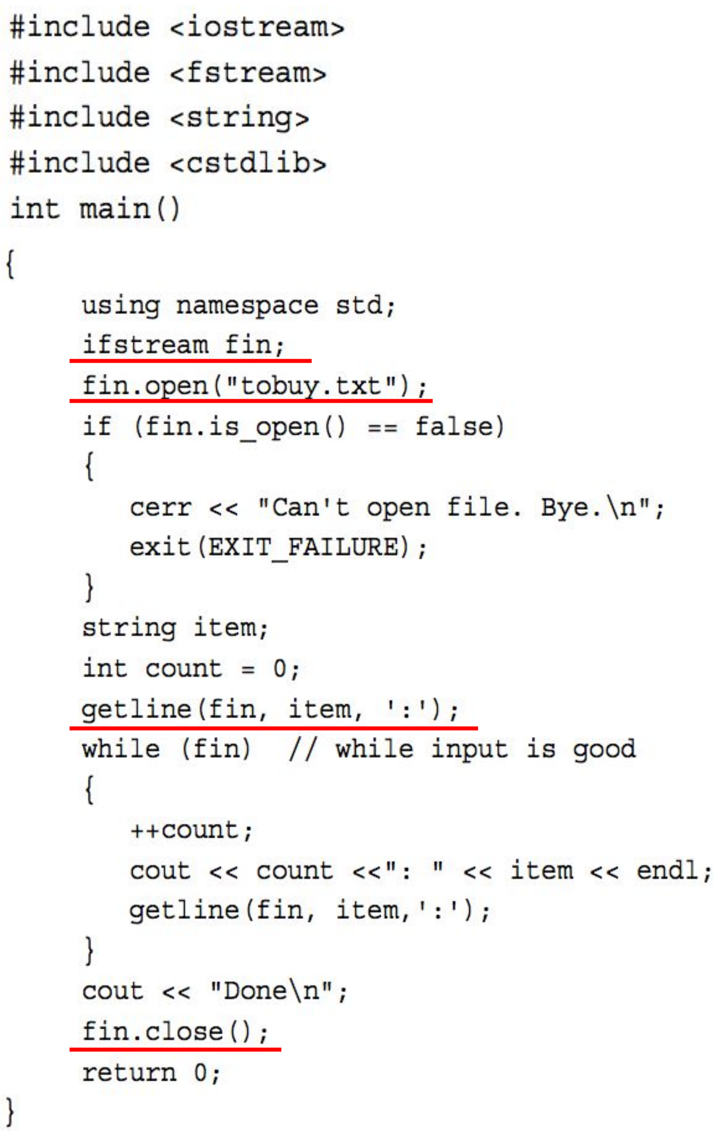
(Recall) std::string - input

- Note that `std::string` automatically resize to the length of target string.



```
char fname[10];
string lname;
cin >> fname;       // could be a problem if input size > 9 characters
cin >> lname;       // can read a very, very long word
cin.getline(fname, 10);  // may truncate input
getline(cin, lname);     // no truncation
```

std::string - input from file



```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
int main()
{
    using namespace std;
    ifstream fin;
    fin.open("tobuy.txt");
    if (fin.is_open() == false)
    {
       cerr << "Can't open file. Bye.\n";
       exit(EXIT_FAILURE);
    }
    string item;
    int count = 0;
    getline(fin, item, ':');
    while (fin)  // while input is good
    {
       ++count;
       cout << count <<": " << item << endl;
       getline(fin, item,':');
    }
    cout << "Done\n";
    fin.close();
    return 0;
}
```

std::string - find

```
size_t find(const string& str, size_t pos = 0) const;
size_t find(char c, size_t pos = 0) const;
[from http://www.cplusplus.com/]
```

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string str("There are two needles in this haystack with needles.");
  string str2("needle");
  size_t found;

  if ((found = str.find(str2)) != string::npos) {
    cout << "first 'needle' found at: " << int(found) << endl;
  }
  str.replace(str.find(str2), str2.length(), "preposition");
  cout << str << endl;
  return 0;
}
```

```
first 'needle' found at: 14
There are two prepositions in this haystack with needles.
```

std::string - substr

```
string substr(size_t pos = 0, size_t n = npos) const;
[from http://www.cplusplus.com/]
```

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string str = "We think in generalities, but we live in details.";
               // quoting Alfred N. Whitehead

  string str2 = str.substr(12, 12);  // "generalities"
  size_t pos = str.find("live");     // position of "live" in str
  string str3 = str.substr(pos);     // get from "live" to the end

  cout << str2 << ' ' << str3 << endl;
}
```

```
generalities live in details.
```

Quiz #3

```
#include <iostream>
#include <map>
using namespace std;

int main() {
  string s = "0123456789";
  size t pos = s.find("345");
  string s2 = s.substr(pos, 5);
  cout << s2 << endl;

  return 0;
}
```

- What is the expected output of this program? (If a compile error is expected, just write down "error").

Next Time

- Next lecture (after the midterm exam):
  - 8 - Inheritance, Const & Class