Using Objects

Object-Oriented Programming with C++

Zhaopeng Cui

Safe way to manipulate strings?

```
std:string
```

The string class

- You must add this at the head of you code

```
#include <string>
```

- Define variable of string like other types

```
string str;
```

- Initialize it w/ string contant

```
string str = "Hello";
```

- Read and write string w/ cin/cout

```
cin >> str;
```

```
cout <<
str;
```

Assignment for string

```
char cstr1[20];
char cstr2[20] = "jaguar" ;

string str1;
string str2 = "panther" ;


cstr1 = cstr2; // illegal
str1 = str2;   // legal
```

Concatenation for string

```
string str3;

str3 = str1 + str2;

str1 += str2;

str1 += "lalala" ;
```

Constructors (ctors)

```
string (const char *cp, int len);
string (const string& s2, int pos);
string (const string& s2, int pos, int len);
```

Sub-string & Search

```
substr (int pos, int len);
```

```
find (const string& s);
```

Modification

```
assign ( . . .);

insert ( . . .);
insert (int pos, const string& s);

erase ( . . .);

append ( . . .);

replace ( . . .);
replace (int pos, int len, const string& s);
```

File I/O

```
#include <fstream>  // dealing with file reading and
writing

std::ofstream File1("test.txt") ;
File1 << "Hello world" << std::endl;

std::ifstream File2("test.txt") ;
std::string str;
File2 >> str;
```

- Assignment 001 on PTA
  - due in Sep. 28 23:59

STL

C++ Standard Library

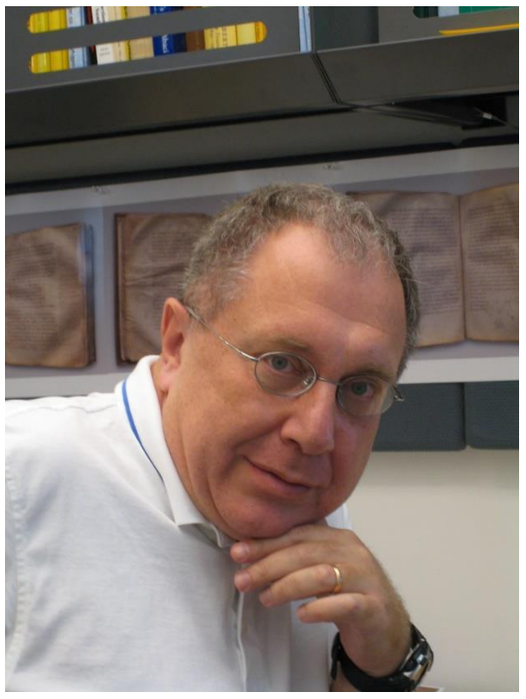
- The C++ standard library provides a wide range of facilities that are usable in standard C++.
  - I/O Stream Library
  - String Library
  - STL
  - Threading Library
  - Mathematical Functions
  - …

- All identifiers in library are in std namespace

using namespace std;

What is STL

- STL = Standard Template Library
- Part of the ISO Standard C++ Library
- Data Structures and algorithms for C++.



Alexander Stepanov

Алекса́ндр Алекса́ндрович Степа́нов

Book:

From Mathematics to Generic Programming

Why should I use STL?

- Reduce development time.
  - Data-structures already written and debugged.
- Code readability
  - Fit more meaningful stuff on one page.
- Robustness
  - STL data structures grow automatically.
- Portable code.
- Maintainable code
- Easy

The three parts of STL

- Containers
  - class templates, common data structures.
- Algorithms
  - Functions that operate on ranges of elements.
- Iterators
  - Generalization of pointers, access elements in a uniform manner.

Containers

- Sequential Containers
- Associative Containers
- Unordered Associative Containers
- Adaptors

Containers

- Sequential Containers
  - array (static), vector (dynamic)
  - deque (double-ended queue)
  - forward_list (singly-linked), list (doubly- linked)
- Associative Containers
- Unordered Associative Containers
- Adaptors

Containers

- Sequential Containers
- Associative Containers
  - set (collection of unique keys)
  - map (collection of key-value pairs)
  - multiset , multimap
- Unordered Associative Containers
- Adaptors

Containers

- Sequential Containers
- Associative Containers
- Unordered Associative Containers
  - hashed by keys
  - unordered_set , unordered_map
  - unordered_multiset , unordered_multimap
- Adaptors

Containers

- Sequential Containers
- Associative Containers
- Unordered Associative Containers
- Adaptors
  - stack, queue, priority_queue, …

Using vector

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

int main() {
vector<int> x;
for (int a = 0 ; a < 1000 ; a++)
    x.push_back(a);

vector<int>::iterator p;
for (p = x.begin(); p < x.end(); p++)
    cout << *p << " " ;
}
```

Basic operations for vector

- Constructors

  vector<Elem> c;

  vector<Elem> c1(c2);

- Simple methods

  V.size( ) **// num items**

  V.empty( ) **// empty?**

  ==, !=, <, >, <=, >=

  v1.swap(v2) **// swap**

- Iterators

  I.begin( ) **// first position**

  I.end( ) **// last position**

- Element access

  V.at(index)

  V[index]

  V.front( ) **// first item**

  V.back( ) **// last item**

- Add/Remove/Find

  V.push_back(e)

  V.pop_back( )

  V.insert(pos, e)

  V.erase(pos)

  V.clear( )

  V.find(first, last, item)

vector

- It is able to increase its internal capacity as required: as more items are added, it simply makes enough room for them.

- It keeps its own private count of how many items it is currently storing. Its size method returns the number of objects currently stored in it.

- It maintains the order of items you insert into it. You can later retrieve them in the same order.

Class Exercises

- Write a code to test vector. Put 5000 items in the vector, and then prints out every fifth element
  - Element 0, element 5, element 10, etc.

Two ways to use vector

- Preallocate

```
vector <int> v(100);
v[80] = 1;  // okay
v[200] = 1;  // bad
```

- Grow tail

```
vector<int> v2;
int i;
while (cin >> i)
    v.push_back(i);
```

Pay attention to efficiency

- Estimate and preserve the memory
- Avoid extra copies

list

- Same basic concepts as vector
  - Constructors
  - Ability to compare lists (==, !=, <, <=, >, >=)
  - Ability to access front and back of list

    x.front(), x.back()
  - Ability to assign items to a list, remove items

    x.push_back(item), x.push_front(item)

    x.pop_back(), x.pop_front()

    x.remove(item)

Using list

- Declare a list of strings
- Add elements
  - –Some to the back
  - –Some to the front
- Iterate through the list
  - –Note the termination condition for our iterator

    p != s.end( )
  - –Cannot use **p < s.end( )** as with vectors, as the list elements may not be stored in order

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

int main() {
    list<string> s;
    s.push_back("hello");
    s.push_back("world");
    s.push_back("stl");

    list<string>::iterator p;
    for (p = s.begin(); p!=
s.end(); p++)
        cout << *p << " ";
    cout << endl;
}
```

Using list

```
#include <iostream>
#include <string>
#include <list>
using namespace std;
int main() {
    list<string> s;
    string t;
    list<string>::iterator p;
    for (int a=0; a<5; a++) {
        cout << "enter a string : " ;
        cin >> t;
        p = s.begin();
        while (p != s.end() && *p < t)
        p++;
        s.insert(p, t);
    }
    for (p=s.begin(); p!=s.end(); p++)
        cout << *p << " ";
    cout << endl;
}
```

map

- Maps are collections that contain pairs of values.
- Pairs consist of a key and a value.
- Lookup works by supplying a key, and retrieving a value.
- An example: a telephone book, a map with strings as keys and values

map<string, string>

name	phone
"Charles Nguyen"	"(531) 9392 4587"
"Lisa Jones"	"(402) 4536 4674"
"William H. Smith"	"(998) 5488 0123"

Using map

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

int main( ) {
    map<string, float> price;
    price["snapple"] = 0.75;
    price["coke"] = 0.50;
    string item;
    double total=0;
    while ( cin >> item )
        total += price[item];
    cout << total << endl;
    return 0;
}
```

Using map

More details here:
https://en.cppreference.com/w/cpp/container/map

Algorithms

Works on a range defined as [first, last).

- for_each, find, count, ...
- copy, fill, transform, replace, rotate, ...
- sort, partial_sort, nth_element, ...
- set_difference, set_union, ...
- min_element, max_element, ...
- accumulate, partial_sum, ...

Iterators

- Connect containers and algorithms.
- Talk about it later
  - after templates and operator overloading.

Pitfalls - access safety

- Accessing an invalid `vector<>` element.

```
vector<int> v;
v[100]=1; // Whoops!
```

Solutions:

- use `push_back()` **for dynamic expansion**
- Preallocate with constructor.
- Reallocate with `resize()`
- Check `size()`

Pitfalls - silent insertion

- Inadvertently inserting into `map<>`

```
if (foo["bob"]==1)
// silently created entry "bob"
```

Use `count()` to check for a key without creating a new entry.

```
if (foo.count("bob"))
```

Or `contains()` introduced in C++20

```
if (foo.contains("bob"))
```

Pitfalls

- Using `empty()` on `list<>`
  - Slow

```
if (my_list.size()== 0)   {...}
```

  - Fast

```
if (my_list.empty())   {...}
```

Other data structures

- set, multiset, multimap
- queue, priority_queue
- stack, deque
- slist, bitset, valarray