Advanced C++

for programming contests

This talk

Using the C++ standard library for programming contests

Prerequisite: Basic C++ (if statements, loops, functions)

Goal: Use existing data structures and algorithms to code faster and with less bugs.

Cheatsheet: `https://soi.ch/s/ws`

About C++

Grader uses C++14. Check your compiler version with `g++ --version`

- GCC 6: Automatically enabled
- GCC 5 or lower: Compile with `-std=c++14` or `-std=c++11`.

Input/Output

Use `cin` and `cout`.

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

int main() {
    int a, b;
    cin >> a >> b;
    int result = a + b;
    cout << result << '\n';
}
```

Example: Sum of n numbers

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

int main() {
  int n;
  cin >> n;
  int sum = 0;
  for (int i=0; i<n; ++i) {
    int x;
    cin >> x;
    sum += x;
  }
  cout << sum << '\n';
}
```

Arrays

Use `std::vector`

```
#include <vector>


vector<int> v(4); // v={0,0,0,0} v.size()==4
v.clear();        // v={}         v.size()==0
v.push_back(4);   // v={4},       v.size()==1
v.push_back(6);   // v={4,6},     v.size()==2
v.push_back(3);   // v={4,6,3},   v.size()==3
v.pop_back();     // v={4,6},     v.size()==2
```

Other lists:

- `vector<string>`
- `vector<vector<int>>`

Example: Read n numbers

```
int n;
cin >> n;
vector<int> v(n);
for (int i=0; i<n; ++i)
  cin >> v[i];
```

Range-based for loop

Iteration via index:

```
int sum = 0;
for (int i=0; i<n; ++i) {
  sum += v[i];
}
```

Iteration via range based for loop:

```
int sum = 0;
for (int x : v) {
  sum += x;
}
```

Why vector

Pro vector:

- Bound-checking can be enabled with `-D_GLIBCXX_DEBUG`
- Integrated interface for algorithms and for loop
- Not slower than raw arrays
- Similar interface as other containers (sets, maps)

Please:

- Do not use arrays
- Do not use `new`/`delete`

Sets

Set = “Menge” (german) or “ensemble” (french)

Elements are always sorted, no duplicates

```
#include <set>

set<int> s;                    // s={},          s.size()==0
s.insert(7);                   // s={7},         s.size()==1
s.insert(3);                   // s={3, 7},      s.size()==2
s.insert(5);                   // s={3, 5, 7},   s.size()==3
s.insert(3);                   // s={3, 5, 7},   s.size()==3
bool has3 = s.count(3);        // true
bool has4 = s.count(4);        // false
s.erase(s.find(3));            // s={7}, s.size()==1
```

Associative Arrays

Key-value mapping, always sorted, has auto-insert

```
#include <map>

map<int, int> m; // m={},                 m.size()==0
m[3] = 1;        // m={3: 1},              m.size()==1
m[7] = 2         // m={3: 1, 7: 2},        m.size()==2
m[3] = 5;        // m={3: 5, 7: 2},        m.size()==2
int x = m[2];    // x=0, m={2: 0, 3: 5, 7: 2}, m.size()==3
```

Duplicates in a set

Q: How to store duplicates in a `set<string>`?

Insert “a”, “b”, “a”, “b”, “c”, “b”, “b”, “a” => 3× a, 4× b, 1× c.

Duplicates in a set

Q: How to store duplicates in a `set<string>`?

Insert “a”, “b”, “a”, “b”, “c”, “b”, “b”, “a” => 3× a, 4× b, 1× c.

A1: `map<string, int>`

insert: `++m[x];`

delete: `if (m[x]==1) m.erase(x); else --m[x];`

Duplicates in a set

Q: How to store duplicates in a `set<string>`?

Insert “a”, “b”, “a”, “b”, “c”, “b”, “b”, “a” => 3× a, 4× b, 1× c.

A1: `map<string, int>`

insert: `++m[x];`

delete: `if (m[x]==1) m.erase(x); else --m[x];`

A2: `multiset<string>`

Containers Overview

- `std::vector<T>`
- `std::map<K,V>`, `std::set<T>`
- `std::deque<T>` (`std::stack<T>`, `std::queue<T>`)
- `std::priority_queue<T>`

Do **not** use: `std::list` (more than 3× slower in almost any case).

Pairs

Pair = two values, “first” and “second”

Type can be different for first and second.

```
pair<int, int> p(3, 4);
auto q = make_pair(3, 4); // same
// p.first == 3
// p.second == 4
p.first = 5; // p == {5, 4}
int a, b;
tie(a, b) = p;
p = make_pair(1, 2);
```

Useful for storing more data per element in a vector.

Alternative to Pairs

Alternative to pair: Custom struct

```
vector<pair<int, int>> graph;

struct edge {
  int to;
  int weight;
};
vector<edge> graph;
```

Chaining pairs

`pair<pair<int, int>, int>` stores 3 values.

`pair<pair<pair<int, int>, int>, int>` stores 4 values.

Please avoid this and use tuples instead (see next slide).

Tuples

- Pair = two values
- Triple = three values
- Quadruple = four values
- Quintuple = five values
- n-Tuple = n values

```
tuple<int, int, int> t(1, 2, 3);
// get<0>(t) == 1
// get<1>(t) == 2
// get<2>(t) == 3
int a, b, c;
tie(a, b, c) = t;
t = make_tuple(4, 5, 6); // t={4, 5, 6}
```

Range-based for loop

For vectors:

```
vector<int> v{3,1,4,1,5};
for (int i : v)
  cout << i << ' ';
```

Range-based for loop

For vectors:

```
vector<int> v{3,1,4,1,5};
for (int i : v)
  cout << i << ' ';
```

Prints:

```
3 1 4 1 5
```

Range-based for loop

For sets:

```
set<int> s{3,1,4,1,5};
for (int i : s)
  cout << i << ' ';
```

Range-based for loop

For sets:

```
set<int> s{3,1,4,1,5};
for (int i : s)
  cout << i << ' ';
```

Prints:

```
1 3 4 5
```

Range-based for loop

For maps:

```
map<int, int> m;
m[3] = 5;
m[7] = 2;
m[1] = 3;
for (auto& p : m) // p.first is the key
                  // p.second is the value
  cout << p.first << ' ' << p.second << ", ";
```

Range-based for loop

For maps:

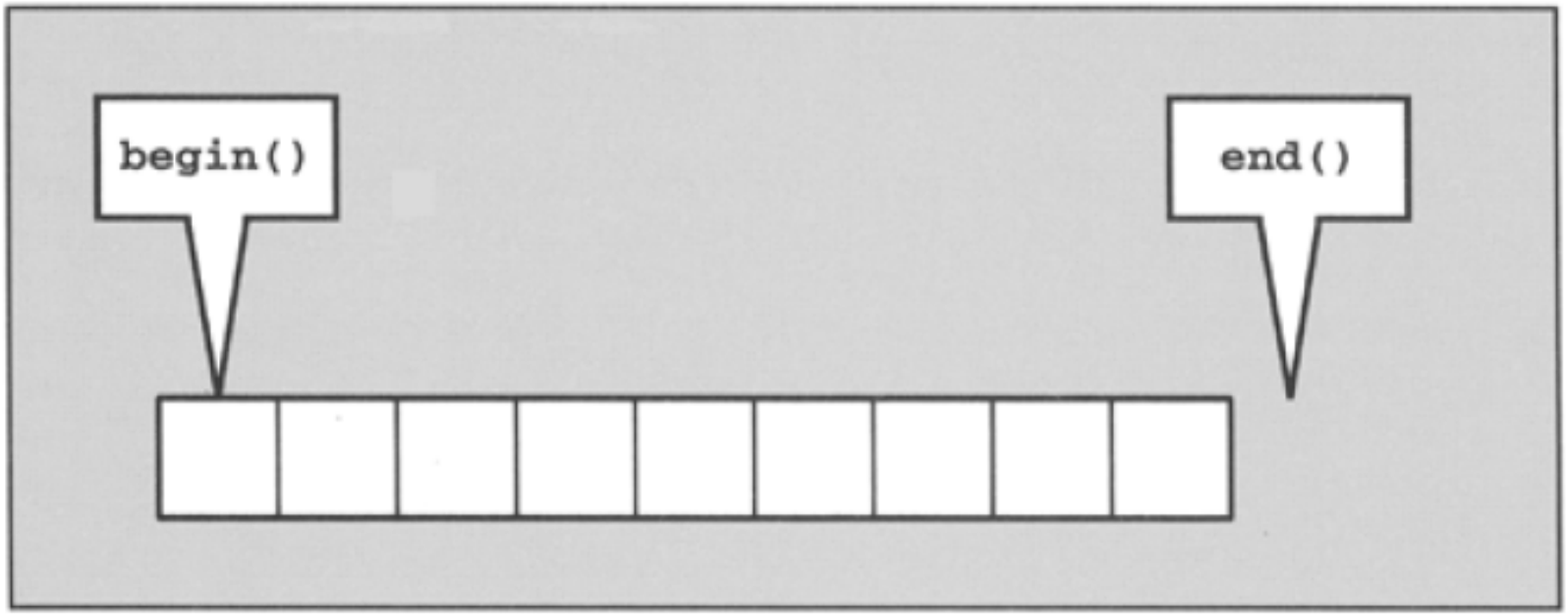
```
map<int, int> m;
m[3] = 5;
m[7] = 2;
m[1] = 3;
for (auto& p : m) // p.first is the key
                  // p.second is the value
  cout << p.first << ' ' << p.second << ", ";
```

Prints:

```
1 3, 3 5, 7 2
```

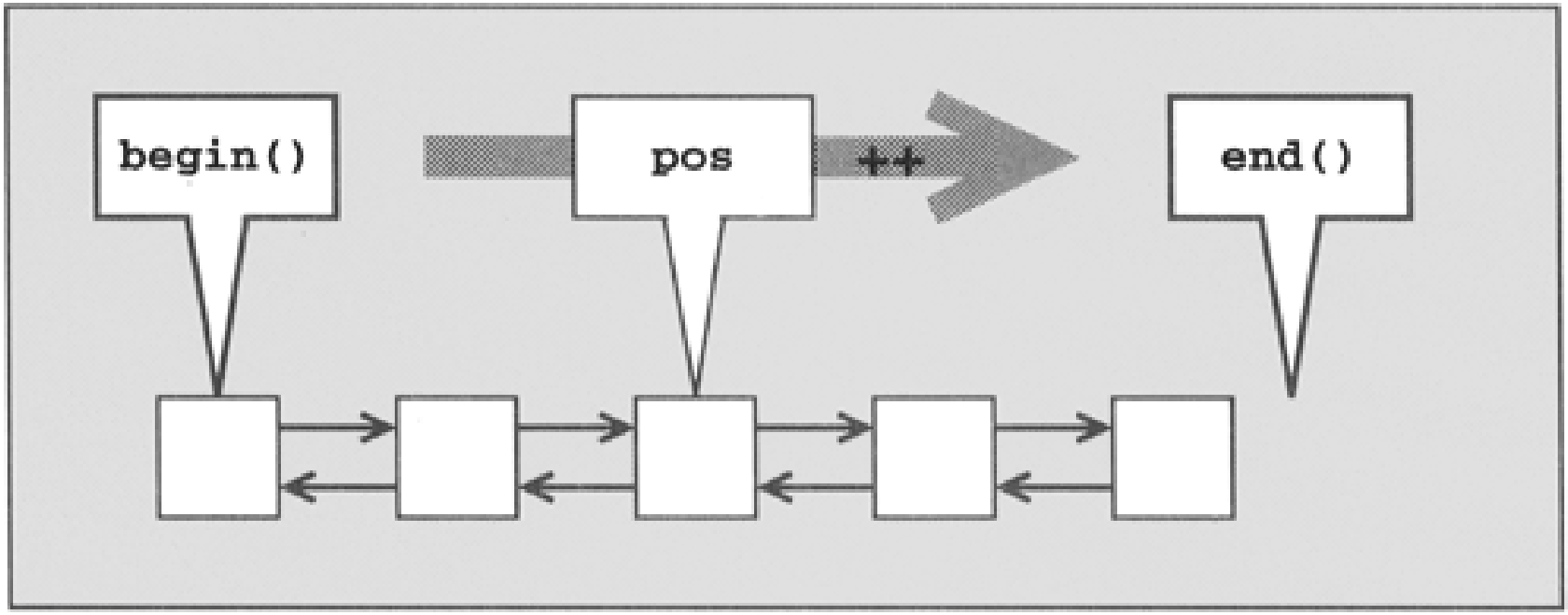
Iterators

- Generalized pointer, not the value but it's position in the container.
- `container.begin()`, `container.end()`



Iterators

- Incrementing: `it++` and `it += n`
- Decrementing: `it--` and `it -= n`



Iterators

- Comparison: `it == it2`, `it != it2`, `it < it2`, `it > it2`, , `it <= it2`, `it >= it2`.
- Getting the value of the element at the given iterator: `*it = x`

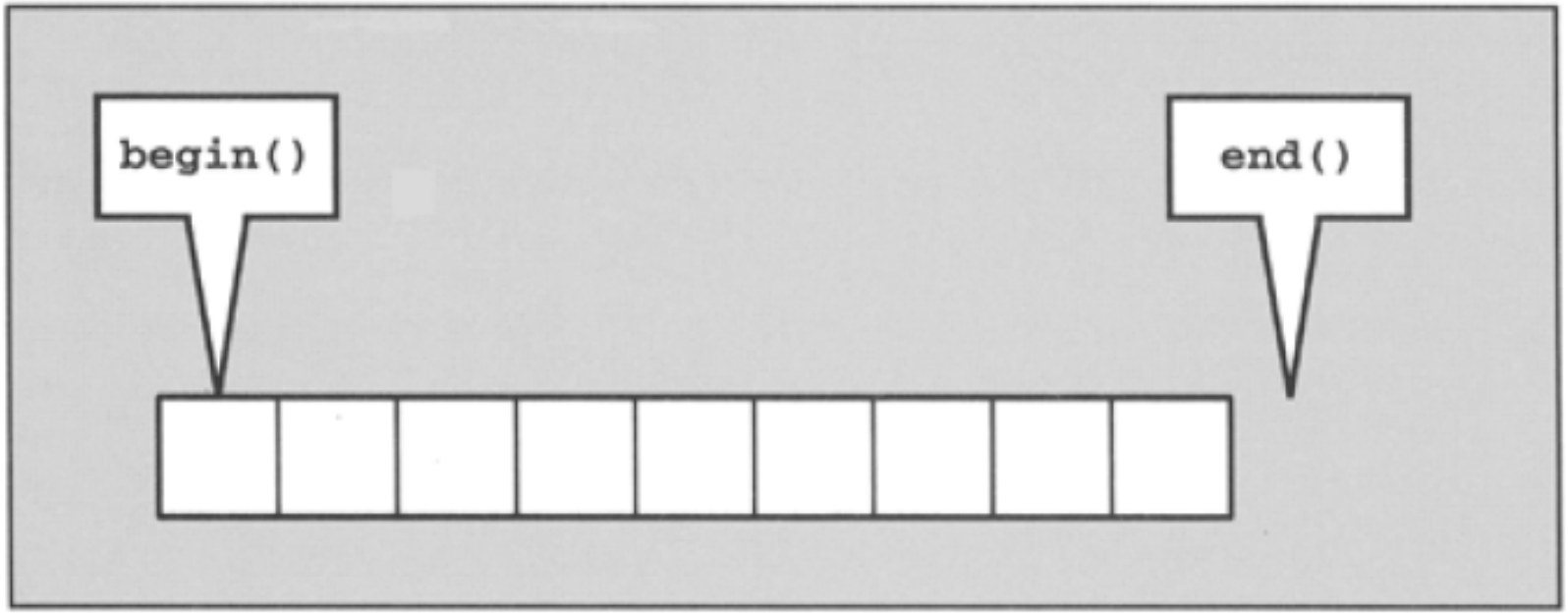
```
vector<int> v{5,2,3};
auto it=v.begin();
*it = 0;    // v={0,2,3}
it += 2;    // it points to 3
*it = 9;    // v={0,2,9}
*--it = 4; // v={0,2,9}
```

Algorithms

Do useful things with iterators!

Use algorithms

Algorithms take two iterators, `first` and `last`. `last` is always exclusive.



Use algorithms

```
vector<int> v{5,4,3,2,1};
sort(v.begin(), v.begin()+2); // v={4,5,3,2,1}
sort(v.begin()+2, v.end());   // v={4,5,1,2,3}
sort(v.begin(), v.end());     // v={1,2,3,4,5}

bool func(int lhs, int rhs) { // lhs = left hand side
  if ((lhs%2) != (rhs%2))
    return (lhs%2) < (rhs%2);
  return (lhs/2) < (rhs/2);
}
sort(v.begin(), v.end(), func);
// v = ?
```

Algorithms

Do useful things with iterators!

- `std::sort`
- `std::binary_search`, `std::lower_bound`, `std::upper_bound`
- `std::unique`
- `std::reverse`
- Doc: `http://en.cppreference.com` (see also `https://soi.ch/s/ws`)

Input/Output revisited

```
#include <iostream>
int n;
cin >> n;
vector<int> v;
for (int i=0; i<n; ++i) {
  int x;
  cin >> x;
  v.push_back(x);
}
cout << n.size() << '\n';
```

Do not slow down your programs:

- Avoid `std::endl`!
- `std::endl`=`'\n'`+`std::flush` and `std::flush` is slow, especially on our grader.

Use I/O Streams

Use the I/O streams instead of `printf` and `scanf`.

- No bugs with wrong flags (`%d` but `long long`)
- Change types easily (`int` -> `long long`)
- Equally fast

Make it even faster:

```
// Turn off synchronization between cin/cout and scanf/pri
ios_base::sync_with_stdio(false);

// Disable automatic flush of cout when reading from cin
cin.tie(0);
```

Terminal Basics

Change directory: `cd <directory>`

Show current directory: `pwd`

Show files in directory (list): `ls`

Compile

```
g++ -Wall -Wextra -g3 -ggdb3 -D_GLIBCXX_DEBUG x.cpp -o x
```

`-Wall -Wextra`: Enable warnings and extra warnings

`-g3 -ggdb3`: Write debug informations for gdb.

`-D_GLIBCXX_DEBUG`: Bound-checking for containers and iterators

If old g++, also add `-std=c++14` or `-std=c++11` or `-std=c++0x` to enable C++14 or C++11 or the predecessor of C++11.

Testing

Input the sample file: `./prog <sample01.in`

Input all sample files:

```
for f in *.in
do
  echo "-- $f --"
  ./prog <$f
done
```

Short:

```
for f in *.in; do echo "-- $f --"; ./prog <$f; done
```

Debugging

In case of a crash use gdb (Gnu DeBugger):

```
$ gdb prog
(gdb) run <sample01.in
(gdb) bt
```

printf-Debugging

Basic idea: Print variables during the whole program flow and figure out when the program crashes or why it behaves weird:

```
for (int i=0; i<10; ++i) {
  cerr << "i=" << i << " sum=" << sum << '\n';
  sum += i;
}
```

Use `cerr` instead of `cout` because `cout` does not flush automatically.

printf-Debugging

Useful macro for debugging:

```
#define DEB(x) cerr << x << '\n'

for (int i=0; i<10; ++i) {
  DEB("i=" << i << " sum=" << sum);
  sum += i;
}
```

Before submitting: remove it at one place

```
#define DEB(x) // cerr << x << '\n'
```

Error prevention

Use `std::assert`.

```
#include <cassert>

void dfs(int n) {
  // n must not be visited
  assert(!graph[n].visited);

  graph[n].visited = true;
  for (int nb : graph[n].adj) {
    if (!graph[nb].visited)
      dfs(nb);
  }
}
```

Include everything

```
#include <bits/stdc++.h>
```

More information

https://soi.ch/s/ws

Use this knowledge to solve those tasks

Category “STL”

- Shop
- Supershop