C++ Features

C++ has a lot of features, some of which are even useful! In addition to having all sorts of language widgets, it also has a sizeable standard library known as the Standard Template Library or STL.

C++ Features

C++ has a lot of features, some of which are even useful! In addition to having all sorts of language widgets, it also has a sizeable standard library known as the Standard Template Library or STL.

- ► `auto`
- ► `vector`
- ► `map` and `unordered_map`
- ► `set` and `unordered_set`
- ► `tuple`
- ► `shared_ptr` and `unique_ptr`

auto (C++ 11)

Compile with `g++ -std=c++11`

```cpp
#include<iostream>
#include<string>
#include<vector>

using namespace std;

int main() {
  auto cstring = "asdf";
  auto str = string("asdf");

  auto thingers = vector<int>();
  return 0;
}
```

"For-each" loops (C++ 11)

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

int main() {
  int nums[] = {1,2,3,4,5,6};

  for(auto i : nums) {
    cout << i * i << endl;
  }

  for(auto i = begin(nums); i != end(nums); i++) {
    cout << (*i) * (*i) << endl;
  }
  return 0;
}
```

Modifying things with "for-each" loops (C++ 11)

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

int main() {
  int nums[] = {1,2,3,4,5,6};

  for(auto& i : nums) {
    i--;
  }

  for(auto i : nums) {
    cout << i * i << endl;
  }
  return 0;
}
```

vector

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

int main() {
  vector<int> v;
  for(int i = 0; i < 10; i++) {
    v.push_back(i);
  }

  v.insert(v.begin() + 4, 200);

  for(vector<int>::iterator it = v.begin(); it != v.end(); i
    cout << *it << endl;
  }
  return 0;
}
```

pair

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

int main() {
  pair<int,int> origin = pair<int,int>(0,0);
  pair<int,int> coord = make_pair(3,5);

  cout << "(" << origin.first << ","
       << origin.second << ")" << endl;
  return 0;
}
```

tuple: a fancier pair (C++ 11)

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

int main() {
  tuple<int,int,string> coord_name(2,4,"A");

  cout << get<2>(coord_name) << ": ("
       << get<0>(coord_name) << ","
       << get<1>(coord_name) << ")\n";

  int x, y;
  tie(x, y, ignore) = coord_name;
  cout << "(" << x << "," << y << ")\n";
  return 0;
}
```

Multiple return from functions, sort of! (C++ 11)

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

tuple<int,int> divide(int divisor, int dividend) {
  return make_pair(divisor / dividend, divisor % dividend);
}

int main() {
  int quotient, remainder;
  tie(quotient, remainder) = divide(13,5);
  cout << "13 / 5 = " << quotient
       << " with remainder " << remainder << endl;

  return 0;
}
```

map

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

int main() {
  map<string, int> ages;
  ages["rick"] = 70;
  ages["morty"] = 14;

  for(auto it = ages.begin(); it != ages.end(); it++) {
    cout << it->first << " is "
         << it->second << " years old.\n";
  }

  return 0;
}
```

set

```cpp
#include<iostream>
#include<set>
using namespace std;
int main() {
  set<int> nums;
  for(int i = 1; i < 10; i++) {
    nums.insert(i);
    nums.insert(i-1);
  }

  if(nums.find(3) != nums.end()) {
    cout << "nums contains 3" << endl;
  }
  if(nums.find(42) == nums.end()) {
    cout << "nums does not contain 42" << endl;
  }
  return 0;
}
```

Smart Pointers (C++ 11)

What are they good for?

- ► Avoiding memory leaks, even in weird edge cases
- ► Stop dereferencing deleted pointers

Smart Pointers (C++ 11)

What are they good for?

- ▶ Avoiding memory leaks, even in weird edge cases
- ▶ Stop dereferencing deleted pointers

Okay, so how do they work?

- ▶ Wrap a pointer inside a class that handles calling delete
- ▶ Describe exactly which objects 'own' the pointer

shared_ptr (C++ 11)

```cpp
#include<iostream>
#include<memory>
using namespace std;
int main() {
    shared_ptr<int> sp(new int);
    *sp = 5;
    shared_ptr<int> sp2(new int(3));

    cout << *sp << " " << *sp2 << endl;
    cout << sp.use_count() << endl;

    sp = sp2;

    cout << *sp << " " << *sp2 << endl;
    cout << sp.use_count() << endl;

    return 0;
}
```

Use-after-free bug

```cpp
#include"list.h"
using namespace std;

Cell<int>* bigger_than(int x) {
  List<int> l;
  for(int i = 0; i < 100; i++) {
    l.append(i);
  }
  Cell<int>* it = l.iterator();
  while(it != NULL && it->elem < x) {
    it = it->next;
  }
  return it; // List's destructor frees this!
}
```

Fixing with shared_ptr

```cpp
#include"list.h"
#include<memory>
using namespace std;

shared_ptr<Cell<int> > bigger_than(int x) {
  List<int> l;
  for(int i = 0; i < 100; i++) {
    l.append(i);
  }
  auto it = l.iterator();
  while(it != NULL && it->elem < x) {
    it = it->next;
  }
  return it; // Not freed by list's destructor!
}
```