Standard library (STL)

Programming Concepts in Scientific Computing

EPFL, Master class

October 25, 2023

STL: input/output streams library: cout/cin

```
#include <iostream>

int main() {
  std::cout << "Hello" << std::endl;
  double a;
  std::cin >> a;
  std::cout << "Hello: " << a << std::endl;
}
```

STL: input/output streams library: cout/cin

```
#include <iomanip>
#include <iostream>

int main() {
  double a;
  std::cin >> a;

  std::cout << std::scientific;
  std::cout << std::setprecision(15);
  std::cout << "Hello: " << a << std::endl;
}
```

STL: string of character: string

```
#include <iostream>
#include <string>

int main(int argc, char *argv[]) {
  std::string name;
  name = argv[1];

  std::cout << "Hello: " << name << std::endl;
}
```

STL: file input/output: fstream

```
#include <fstream>

int main(int argc, char *argv[]) {
  std::string name;
  name = argv[1];
  std::ifstream fin(name.c_str());
  while (fin.good()) {
    double a;
    fin >> a;
  }
}
```

STL: string input/output: stringstream

```
#include <iostream>
#include <sstream>

int main(int argc, char *argv[]) {
  std::stringstream sstr;
  for (int i = 1; i < argc; ++i) {
    sstr << argv[i] << ",";
  }
  std::cout << "Comma separated list of arguments: ";
  std::cout << sstr.str() << std::endl;

  int N = 0;
  sstr >> N;
}
```

STL: containers

- STL Containers: generic description (design pattern)
- They are based on a container-iterator relation
- This simplifies the transition from one type to another
- Classical types are:
  - vector
  - list
  - set
  - map
  - multimap

STL containers: vector

The include:

```
#include <vector>
```

STL containers: vector

Loops:

```
int N = 100;
std::vector<double> v(N);

for (unsigned int i = 0; i < v.size(); ++i) {
  v[i] = 100.;
}
```

Templated class

What does this 'unsigned' keyword mean?

STL containers: vector

Loops:

```
int N = 100;
std::vector<double> v(N);

for (unsigned int i = 0; i < v.size(); ++i) {
  v[i] = 100.;
}
```

Or equivalently

pointer

```
std::vector<double>::iterator it = v.begin();
std::vector<double>::iterator end = v.end();

while (it != end) {
  *it = 100;
  ++it;
}
```

STL containers: map

Associating keys with values Python: dictionary

```
#include <map>
```

STL containers: map

Example with planets

```
enum ParticleType {
  planet,
  star,
  atom,
};

  std::map<std::string, ParticleType> particle_types;

  particle_types["mars"] = planet;
  particle_types["sun"] = star;
  particle_types["copper"] = atom;
```

STL containers: map
Looping over keys and values

```
std::map<std::string, ParticleType>::iterator it = partic
std::map<std::string, ParticleType>::iterator end = parti

for (; it != end; ++it) {
  std::string key = it->first;
  ParticleType p_type = it->second;

  std::cout << "key:" << key << " -> ";
  std::cout << p_type << std::endl;
}
```

STL: algorithms

There are many STL algorithms available.

Examples:

- std::find searches for a match in a container in $O(N)$ advances.
- std::sort sorts a container in $O(N \cdot log_2(N))$ swaps (which then is not the same for list, vectors)
- std::binary_search searches into a sorted container in $O(log2(N))$ advances (which is then long for non random access iterators like lists)

Take away message

http://en.cppreference.com/

http://www.cplusplus.com/reference/