Template and Containers

Seonghyun Park

dd December, 2018

Outline

1. Template
   - Motivation
   - Syntax
2. Containers
   - Motivation
   - Example

Today's Goal

- Understand usage of templates (template classes and template functions)
- Learn how to use C++ Standard Template Library (STL) containers to your own!

Motivation

- Think about linked list we have implemented so far.
- It's `Node` contains an `int` value.
- What if you need a list of `char` or `std::string`?
- Just copy and paste your code and change types? or rely on the black magic of `void *`?
- Isn't it too tedious, inefficient or unsafe?

Syntax

- In C++ we use template to declare generic functions or classes.

```
template <typenae T>
void swap(T& t, T& u)
{
  T tmp = t;
  t = u;
  u = tmp;
}
```

Syntax

```
template <typenae T, U>
class Node {
  T t; U u;
  ...
};

template <typename T, U>
class List {
  Node<T, U> head;
  ...
};
```

Motivation

- As we've seen so far, most programs involves creating collections of values and then manipulating such collections.
- Even creating a `string` and manipulating characters from that `string` can be seen as actions of these categories.
- We call data structures, whose main purpose is to hold data and perform manipulations, as *containers*.
- C++ provides programmers with a huge set of container classes and operator primitives.

Example

- `vector`, `set`, `list`, `unordered_map`
- All STL containers come with easy-to-use interfaces.
- See `cppreference` for further information.