Standard Template Library (STL) is C++’s library of useful data structures & algorithms

- Like java.util/java.lang
- Like Python sets, dictionaries, collections

Templates are covered in detail later in the course; we’ll give them a quick look now
Templates are a way of writing an object (Node) or function (print_list) to work with any type. You simultaneously define a family of related objects/functions.
```cpp
struct Node {
    T payload; // 'T' is placeholder for a type
    Node *next;
};

void print_list(Node *head) {
    Node *cur = head;
    while (cur != NULL) {
        cout << cur->payload << " ";
        cur = cur->next;
    }
    cout << endl;
}
```

We could replace T with int, float, char, or std::string and this would compile & work
template<typename T>
struct Node {
  T payload;
  Node *next;
};

template<typename T>
void print_list(Node<T> *head) {
  Node<T> *cur = head;
  while (cur != NULL) {
    cout << cur->payload << " ";
    cur = cur->next;
  }
  cout << endl;
}

Same example, using templates

One struct/function, works for (almost) any type T
```cpp
int main() {
    Node<float> f3 = {95.1f, NULL};  // float payload
    Node<float> f2 = {48.7f, &f3};   // float payload
    Node<float> f1 = {24.3f, &f2};   // float payload
    print_list(&f1);

    Node<int> i2 = {239, NULL};     // int payload
    Node<int> i1 = {114, &i2};      // int payload
    print_list(&i1);

    return 0;
}
```

```
g++ -c ll_template_cpp.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ll_template_cpp ll_template_cpp.o
$ ./ll_template_cpp
24.3 48.7 95.1
114 239
```
With STL we use types like `vector<string>`

We read that type as “a vector of strings”

- `vector<string>` – a vector of `std::string`s
- `vector<float>` – a vector of floats
- `map<string, int>` – a structure that maps `std::string`s to `ints`

Similar to Java generics