601.220 Intermediate Programming

Spring 2023, Day 31 (April 10th)

Today's agenda

- Review exercise 30
- Day 31 recap questions
- Exercise 31

Reminders/Announcements

- HW7 is due **Friday** by 11 pm
- Final project team formation
  - Submit Google form by **11 am tomorrow** (Tuesday, April 11th)
  - See Piazza post 571 (pinned)

Exercise 30 review

```
// copy constructor
int_set::int_set(const int_set& orig)
  : head(nullptr), size(0) {
  *this = orig;
}

// destructor
int_set::~int_set() {
  clear();
}
```

assignment

Exercise 30 review

```
// += operator
int_set& int_set::operator+=(int new_value) {
  add(new_value);
  return *this;
}
```

Exercise 30 review

```
// assignment operator
int_set& int_set::operator=(const int_set &rhs) {
  if (this != &rhs) {
    clear(); // delete old linked list

    int_node *n = other.head;
    while (n != nulptr) {
      add(n->get_data());
      n = n->get_next();
    }
  }
  return *this;
}
```

Note: inefficient because add is $O(N)$. Overall running time is $O(N^2)$.

Exercise 30 review

(cout << a) << b;

```
// output stream insertion operator
std::ostream& operator<<(std::ostream& os, const int_set& s){
  int_node *n = s.head;
  os << "{";
  while (n != nullptr) {
    os << n->get_data();
    if (n->get_next() != nullptr) { os << ", "; }
    n = n->get_next();
  }
  os << "}";
  return os;
}
```

Day 31 recap questions

1. How do we declare a template function?
2. Under what conditions would you consider making a function templated?
3. What is template instantiation?
4. Can we separate declaration and definition when using templates?
5. Why shouldn't template definitions be in .cpp files?

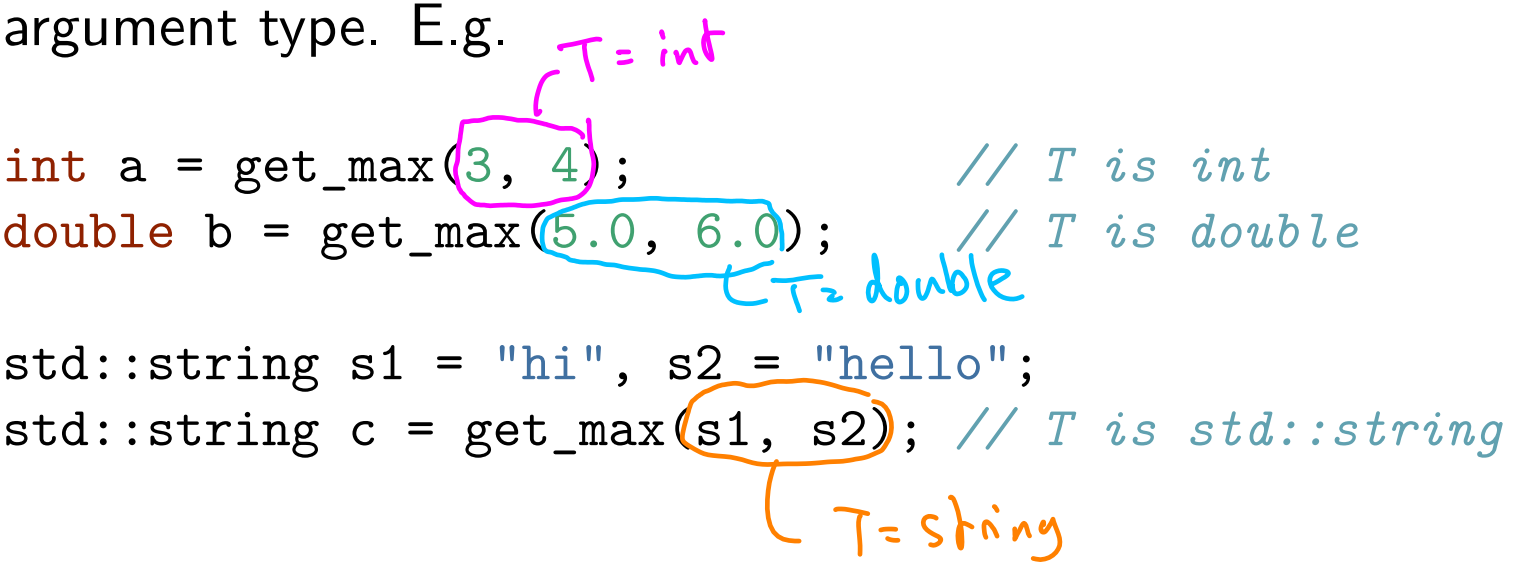
1. How do we declare a template function?

type parameter

```
// Example
template<typename T>
T get_max(const T &left, const T &right) {
  if (left > right) { return left; }
  else              { return right; }
}
```

Type parameter inference

T is a "type parameter". In a call to get_max, T is inferred from the argument type. E.g.




```
int a = get_max(3, 4);              // T is int
double b = get_max(5.0, 6.0);       // T is double

std::string s1 = "hi", s2 = "hello";
std::string c = get_max(s1, s2); // T is std::string
```

2. Under what conditions would you consider making a function templated?

Template functions are ~~usuefful~~ useful when you want to allow the function to work with a variety of different data types.

The data types that will be substituted for the type parameter(s) must have common operations that will be used by the template function.

For example, the `get_max` function (shown previously) requires the data type `T` to have a `>` operator, and also a copy constructor.

All of the built-in types (`int`, `double`, etc.) have these a `>` operator, assignment operator, and copy constuctor. (A.k.a. "value semantics".)

3. What is template instantiation?

Template instantiation is the substitution of actual data types for type parameters. For example, in

```
int a = get_max(3, 4);
```

T=int

the type `int` is substituted for `T`. So:

```
template<typename T>
T get_max(const T &left, const T &right) {
  if (left > right) { return left; }
  else              { return right; }
}
```

replace T with int

```
// instantiated with T=int
int get_max(const int &left, const int &right) {
  if (left > right) { return left; }
  else              { return right; }
}
```

4. Can we separate declaration and definition when using templates?

foo.h

```
int get_max( int a, int b );
```

It is possible to separate the declaration and definition of template functions and classes.

However, this is more complicated and less flexible than just putting the definition of the template function or class in a header file.

The compiler doesn't normally instantiate a teemplate function or class until the function or class is actually used. To instantiate the function or class, the compiler needs the definition.

So, we generally put the definitions for template functions and classes directly in a header file.

5. Why shouldn't template definitions be in .cpp files?

Template functions (including member functions of template classes) can't be compiled into machine code and put into an object (`.o`) file.

The basic problem is that there are an infinite number of possible ways a template function or class could be instantiated. For example, `vector<int>`, `vector<string>`, `vector<YourClass>`, etc. The compiler doesn't know which instantiations your program will need until it actually sees the code that uses `vector`, and knows which types will be substituted for `vector`'s type parameter.

".inc" files

One way to allow template function definitions to be separated from a template class declaration, but still be available from a header file, is to use an ".inc" file. The .inc file contains the definitions of member functions of the template class. The header file would look like this:

```
#ifndef MY_SET_H
#define MY_SET_H

template<typename T> class my_set {
  // declarations of member functions of my_set<T>
};

#include "my_set.inc"

#endif // MY_SET_H
```

Exercise 31

- Convert your `int_set` class from Exercise 30 to a template class
- Note that `ex30-sol` (Exercise 30 reference solution) has been added to the public repo, `git pull` to get it
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes