Day 30 (Mon 04/11)

- Exercise 30 review
- Day 31 recap questions
- Exercise 31

Announcements/reminders

- HW6 due Wednesday 4/13
  - written homework, no late submissions

- HW7 due Wednesday, 4/20

- final project: details soon

Exercise 30

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

assign op

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

Exercise 30

```
// += operator
int_set& int_set::operator+=(int new_value) {
  add(new_value);
  return *this;    //for consistency - assignment ops return the value assigned
}

// assignment operator
int_set& int_set::operator=(const int_set& other) {
  if (this != &other) {
    int_node *n = other.head;
    while (n != nullptr) {
      add(n->get_data());
      n = n->get_next();
    }
  }
  return *this;
}
```

Exercise 30

```
// 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?

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

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

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

Template functions are 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 can be used by the template function.

For example, the get_max function on the previous slide requires the data type substituted for T to have a > operator, and also a copy constructor.

All of the built-in types (int, double, etc.) have a default constructor, assignment operator, and copy constructor.

3. What is template instantiation?

Template instantiation is the substitution of actual specific data types for type parameters. For example:

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

The type int is substituted for T. So:

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

```
// 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?

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 normally doesn't instantiate a template 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 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. E.g., vector<int>, vector<string>, vector<YourClass>, etc. The compiler doesn't know which ones your program will need, and wouldn't necessarily know about the data types that will be substituted for type parameters.

".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 {
public:
  // declarations of member functions of my_set
};

#include "my_set.inc" // the .inc file has the definitions of the member functions

#endif // MY_SET_H
```