CS11600: Introduction to Computer Programming (C++)

Lecture 12

Svetlozar Nestorov
University of Chicago

Outline

- Robust class design
- Template functions:
  - Motivation
  - Instantiation
  - Overloading
  - Specialization
- Friend functions

Robust Class Design

- Implement two private member functions copy and `free` that handle dynamic memory allocation and deallocation for the class.

```
class ClassName {
    void copy(const ClassName &);
    void free();
    ~ }
```

- Call these functions in constructors, destructor, and assignments.

Generic Functions

- Find max of two numbers:

```
x > y ? x : y
```

- Works for int, double, char*, etc.
- But, consider a `max` function:

```
int max(int x, int y) {
   return x > y ? x : y;
}
```

- Does it work for double, char*?

Template Functions

- Idea: make type an implicit parameter.

```
template <class T>
T & max(T & x, T & y) {
   return x > y ? x : y;
}
```

- May have more than one template class:

```
template <class T1, class T2, …>
```


Instantiation

- Compiler instantiate the correct function based on parameter types:

```
int a = 3, b = 4;
max(a, b);
```

instantiates to:

```
int & max(int & x, int & y) {
   return x > y ? x : y;
}
```

Overloading

- Different template function with the same name but *different* signatures.

```
template <class T>
T & max(T *array, int size);
```


Function Call Resolution

- When resolving a function call the compiler chooses in the following order:
  1. Non-template function with exact argument match.
  2. Template function with exact argument match.
  3. Non-template function with argument conversion.

Specialization

- Implement a function with concrete argument types.
- This function takes precedence over the template function.

```
double & max(double & x, double & y);
```

will be called when arguments are two doubles, not the template function.

Friend Functions

- Give functions direct access to private members of a class:

```
class ClassName {
   …
   friend Type func(signature);
   …
}
```