CS 162
Intro to Computer Science II

Templates
Function Templates

- **Function template**: A pattern for creating definitions of functions that differ only in the type of data they manipulate. It is a generic function.

- They are better than overloaded functions, since the code defining the algorithm of the function is only written once.
Example

Two functions that differ only in the type of the data they manipulate

```c
void swap(int &x, int &y)
{
    int temp = x; x = y;
    y = temp;
}

void swap(char &x, char &y)
{
    char temp = x; x = y;
    y = temp;
}
```
A swap Template

The logic of both functions can be captured with one template function definition

```cpp
template<class T>
void swap(T &x, T &y)
{
    T temp = x; x = y;
    y = temp;
}
```
Using a Template Function

• When a function defined by a template is called, the compiler creates the actual definition from the template by inferring the type of the type parameters from the arguments in the call:

```cpp
int i = 1, j = 2;
swap(i, j);
```

• This code makes the compiler instantiate the template with type `int` in place of the type parameter `T`
Function Template Notes

• A function template is a pattern
• No actual code is generated until the function named in the template is called
• A function template uses no memory
• When passing a class object to a function template, ensure that all operators referred to in the template are defined or overloaded in the class definition
Function Template Notes

• All data types specified in template prefix must be used in template definition.

• Function calls must pass parameters for all data types specified in the template prefix.

• Function templates can be overloaded – need different parameter lists.

• Like regular functions, function templates must be defined before being called.
Where to Start When Defining Templates

- Templates are often appropriate for multiple functions that perform the same task with different parameter data types.
- Develop function using usual data types first, then convert to a template:
  - add template prefix
  - convert data type names in the function to a type parameter (i.e., a T type) in the template.
Class Templates

• It is possible to define templates for classes. Such classes define abstract data types

• Unlike functions, a class template is instantiated by supplying the type name (int, float, string, etc.) at object definition
Consider the following classes

1. Class used to join two integers by adding them:
   
   ```
   class Joiner
   { public:
       int combine(int x, int y)
       { return x + y; }
   };
   ```

2. Class used to join two strings by concatenating them:
   
   ```
   class Joiner
   { public:
       string combine(string x, string y)
       { return x + y; }
   };
   ```
Example class Template

A single class template can capture the logic of both classes: it is written with a template prefix that specifies the data type parameters:

template <class T>
class Joiner
{
  public:
    T combine(T x, T y)
    {
      return x + y;
    }
};
Using Class Templates

To create an object of a class defined by a template, specify the actual parameters for the formal data types

```cpp
Joiner<double> jd;
Joiner<string> sd;
cout << jd.combine(3.0, 5.0);
cout << sd.combine("Hi ", "Ho");
```

Prints 8.0 and Hi Ho
Class Templates and Inheritance

• Templates can be combined with inheritance

• You can derive
  – Non template classes from a template class: instantiate the base class template and then inherit from it
  – Template class from a template class

• Other combinations are possible
Template Design Strategy

• Develop function with actual data types
• Test and debug this function
• Add the template
  – Replace type names with type parameter from the template
• Advantages:
  – Develop a base or specific case
  – Develop the algorithm first
  – Then add the template syntax
Template Design Considerations

• Can use any type in template for which the algorithm behaves in an appropriate way

• e.g., swapValues() template function
  – Cannot use type for which assignment operator isn’t defined
  – Example: an array:
    ```
    int a[10], b[10];
    swapValues(a, b);
    ```
  – Arrays cannot be "assigned"!