C++ Data Structures
Prerequisites

CHAPTER 3: C++ TEMPLATE FUNCTIONS

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

- Understand what is Generic Function
- Implementation of Template Functions

LECTURE 1

Generic Function

Finding the Maximum of Two **Integers**

- Here's a small function that you might write to find the maximum of two integers.

```
int maximum(int a, int b)
{
      if (a > b)
          return a;
      else
          return b;
}
```

Finding the Maximum of Two **Doubles**

- Here's a small function that you might write to find the maximum of two **double** numbers.

```
int maximum(double a, double b)
{
      if (a > b)
          return a;
      else
          return b;
}
```

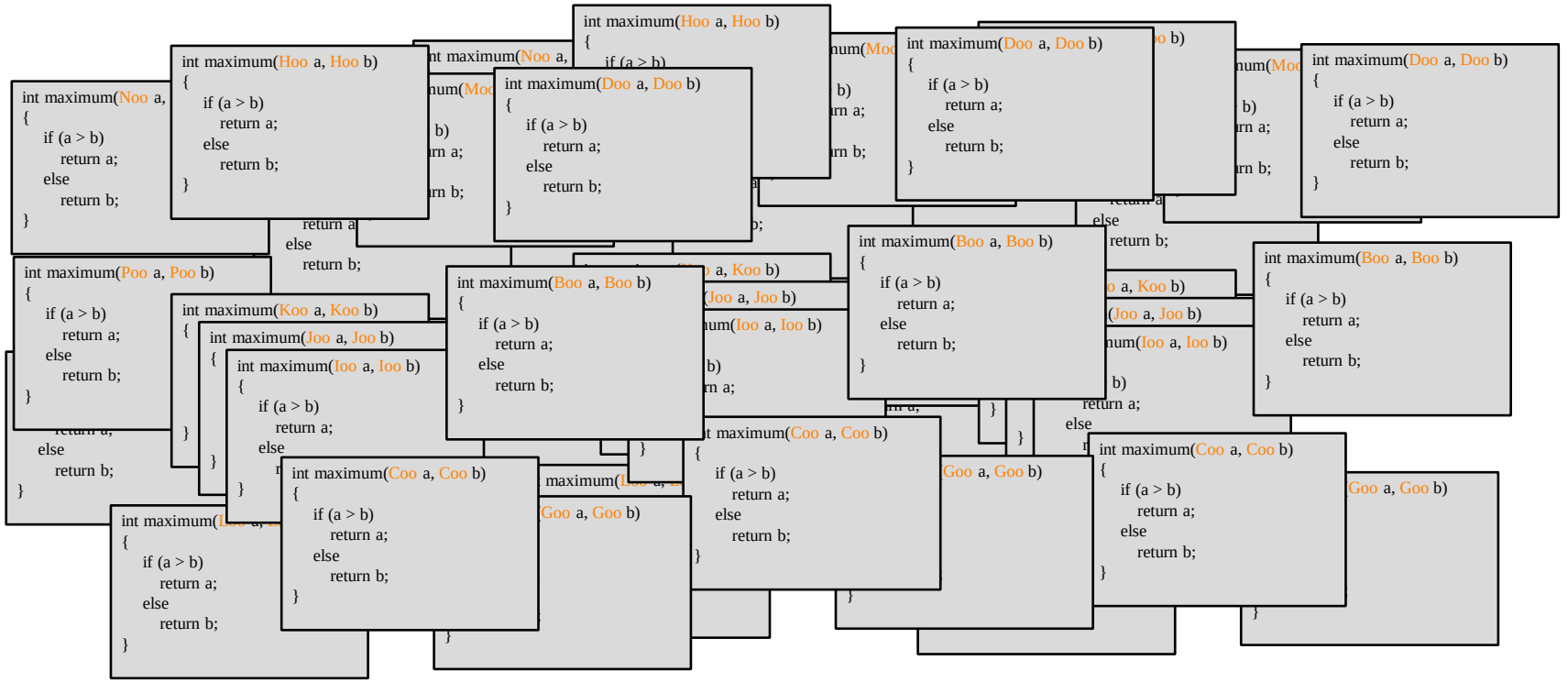
Finding the Maximum of Two **Knafns**

- Here's a small function that you might write to find the maximum of two **knafns**.

```
int maximum(knafn a, knafn b)
{
      if (a > b)
          return a;
      else
          return b;
}
```

One Hundred Million Functions…

- Suppose your program uses 100,000,000 different data types, and you need a maximum function for each…



A Template Function for Maximum

- This template function can be used with many data types.

```
template <class Item>
Item maximum(Item a, Item b)
{
      if (a > b)
          return a;
      else
          return b;
}
```

A Template Function for Maximum

- When you write a template function, you choose a data type for the function to depend upon...

```
template <class Item>
Item maximum(Item a, Item b)
{
      if (a > b)
          return a;
      else
          return b;
}
```

A Template Function for Maximum

- A template prefix is also needed immediately before the function's implementation:

```
template <class Item>
Item maximum(Item a, Item b)
{
      if (a > b)
          return a;
      else
          return b;
}
```

Using a Template Function

- Once a template function is defined, it may be used with any adequate data type in your program...

```
template <class Item>
Item maximum(Item a, Item b)
{
      if (a > b)
          return a;
      else
          return b;
}
```

```
cout << maximum(1,2);
cout << maximum(1.3, 0.9);
...
```

Finding the Maximum Item in an Array

- Here's another function that can be made more general by changing it to a template function:

```
int array_max(int data[  ], size_t n)
{
     size_t i;
     int answer;

     assert(n > 0);
     answer = data[0];
     for (i = 1; i < n; i++)
          if (data[i] > answer) answer = data[i];
     return answer;
}
```

Finding the Maximum Item in an Array

- Here's the template function:

```
template <class Item>
Item array_max(Item data[], size_t n)
{
     size_t i;
     Item answer;

     assert(n > 0);
     answer = data[0];
     for (i = 1; i < n; i++)
          if (data[i] > answer) answer = data[i];
     return answer;
}
```

LECTURE 2

Implementation

Namespace std

- Using namespace std can sometime be dangerous.
- Std has min function, max function, so it may work.

template1.cpp

```
#include <iostream>

using namespace std;

int min(int x, int y){
    return (x<y) ? x : y;
}

int main(int argc, const char* arvg[]){
    cout << min(2, 4) << endl;
    cout << min(5, 3) << endl;
    cout << min(2.2, 4.4) << endl;
    cout << min(5.5, 3.3) << endl;
    return 0;
}
```

```
2
3
2.2
3.3
```

template2.cpp

```cpp
#include <iostream>

using namespace std;
double small(double x, double y){
    return (x<y) ? x : y;
}
int small(int x, int y){
    return (x<y) ? x : y;
}

int main(int argc, const char* arvg[]){
    cout << small(2, 4) << endl;
    cout << small(5, 3) << endl;
    cout << small(2.2, 4.4) << endl;
    cout << small(5.5, 3.3) << endl;
    return 0;
}
```

```
2
3
2.2
3.3
```

template3.cpp

```cpp
#include <iostream>

using namespace std;
template<typename T>
T small(T x, T y){
    return (x<y) ? x : y;
}

int main(int argc, const char* arvg[]){
    cout << small(2, 4) << endl;
    cout << small(5, 3) << endl;
    cout << small(2.2, 4.4) << endl;
    cout << small(5.5, 3.3) << endl;
    string a = "apple", b = "banana";
    cout << small(a, b) << endl;
    cout << small(b, a) << endl;
    return 0;
}
```

```
2
3
2.2
3.3
apple
apple
```

template<T x, T y>

- Declaration of a template function (Generic Function)
- The function must be right below this template statement.

```
#ifndef BIG_SMALL_H
#define BIG_SMALL_H
template<typename T>
T big(T x, T y){
   return (x>y) ? x : y;
}

template<typename T>
T small(T x, T y){
   return (x<y) ? x : y;
}
#endif

#include <iostream>
#include "bigsmall.h"
using namespace std;

int main(int argc, const char* arvg[]){
    cout << small(2, 4) << endl;
    cout << small(5, 3) << endl;
    cout << small(2.2, 4.4) << endl;
    cout << small(5.5, 3.3) << endl;
    cout << big("apple", "small") << endl;
    return 0;
}
```

template4.cpp

```
2
3
2.2
3.3
apple
```