Introduction to Templates

Overview

- Introduction to genericity
- Templates
- Creating templates
- Using templates

2

This material is intended for students of the "C++ Programming for Financial Engineering" certificate
A joint project by the Baruch MFE program, Datasim Education BV and Quant Network LLC
© Datasim Education BV 2011. Unauthorized distribution is prohibited.

Introduction to Genericity

- The C++ construction for genericity == templates
- Templates are descriptors for classes
- Typical classes include
- bags maps queues
- sets stacks symbol_tables
- list trees sorted_arrays
- arrays matrices

3

Templates

- Template class is a description of the class
- Class is instance of template
- Works with generic types instead of concrete types

4

Creating Templates: General Rules

- Header and code files as with non-template classes
- We need to identify underlying data type(s)
- Combination of class name + type is important
- New keyword 'template'

5

Template Specifier

- Use the template specifier for a generic scope
- template <class Type> means next scope is generic
  - Members functions have their own scope
- Nowadays typename instead of class is preferred
  - template <typename Type>

6

This material is intended for students of the "C++ Programming for Financial Engineering" certificate
A joint project by the Baruch MFE program, Datasim Education BV and Quant Network LLC
© Datasim Education BV 2011. Unauthorized distribution is prohibited.

Example
Array State

```
template <typename Type>
class Array
{ // Declaration and definition of a class template

private:
  Type* arr;
  int sz;

public:

};
```

7

When to Use Type

- The template arguments (type) become part of the class name
- Function names have no type
  - Constructors and destructor are functions
- Function with template class as input/output argument must have type
  - Input or output do not have to be the same as the current

8

Type Usage

```
template <typename Type>
class Array
{
public:
  // Distinguishing between function names and class names

  // Function name
  Array();

  // Class name
  Array(const Array<Type>& source);
};
```

9

Hints and Remarks

- Choose good names for underlying types
- Template class name == class name + type

10

This material is intended for students of the "C++ Programming for Financial Engineering" certificate
A joint project by the Baruch MFE program, Datasim Education BV and Quant Network LLC
© Datasim Education BV 2011. Unauthorized distribution is prohibited.

Source File : General Rule on Syntax

1. Keyword ‘template’ and underlying data types
2. Return type
3. (Template class name including type)::(function name)
4. Input arguments

11

The Template Source File

- Where you refer to the class use class_name<Type>

```
template <typename Type>
ARRAY<Type>::ARRAY()
{
  //...
}

template <typename Type>
ARRAY<Type>& ARRAY<Type>::operator = (const ARRAY<Type>& arr)
{
  //...
}
```

12

Using Templates in Source

- Include source file
  - `#include "array.cpp"`
- Create class and object by substituting the generic type

```
void main()
{
  ARRAY<int> intarray(10);
  ARRAY<double> dblarray(20);
}
```

13

Including Source Files

- Template source file is included
- To prevent multiple inclusion create a #ifndef structure
- Sometimes template is not separated in different files
- Template is placed in single file with extension ".tpp" or just ".h"

14

This material is intended for students of the "C++ Programming for Financial Engineering" certificate
A joint project by the Baruch MFE program, Datasim Education BV and Quant Network LLC
© Datasim Education BV 2011. Unauthorized distribution is prohibited.

Generic Functions

- Instead of class for several types, function for several types

```
template <typename T>
void swap(T& t1, T& t2)
{
  T tmp = t1;
  t1 = t2;
  t2 = tmp;
}
```

15

Tips For Creating Templates

- Templates should advertise what they expect of the formal data types (especially operators)
- Templates should provide minimal functionality; grow your own specialisation's

16

Practicalities

- When creating template classes, you choose between inline code (1 file) or separate header or source file
- As new C++ developer you will probably use template classes rather than create your own
- Most compiler errors caused by syntax omissions (can be frustrating...)
- Flag the assumptions made by the template code on the underlying types

17

This material is intended for students of the "C++ Programming for Financial Engineering" certificate
A joint project by the Baruch MFE program, Datasim Education BV and Quant Network LLC
© Datasim Education BV 2011. Unauthorized distribution is prohibited.