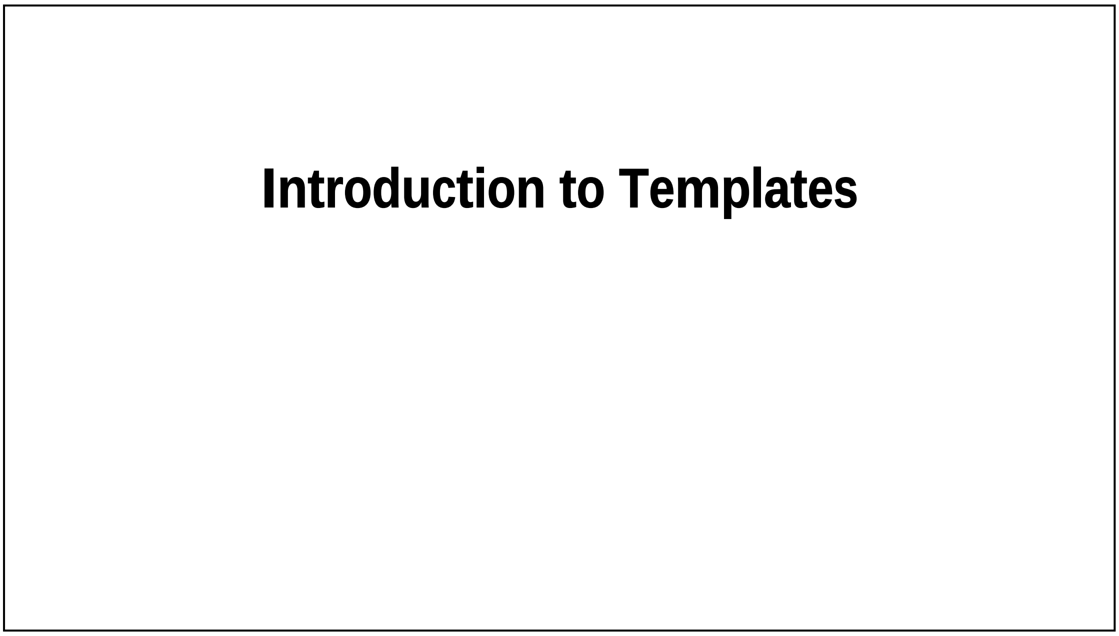
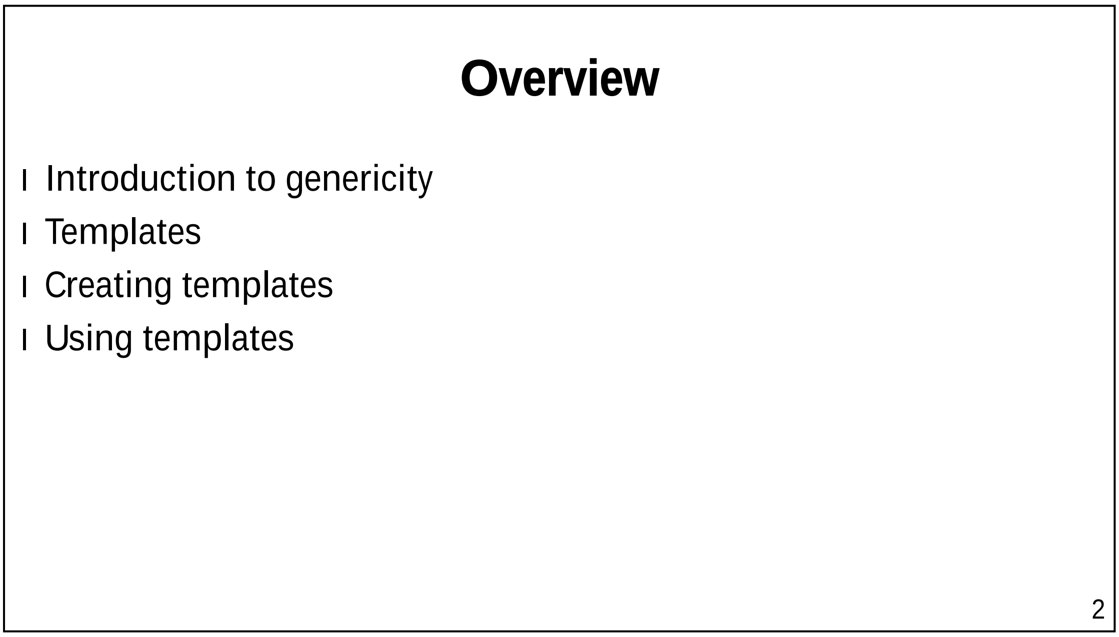


Introduction to Templates



Overview

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

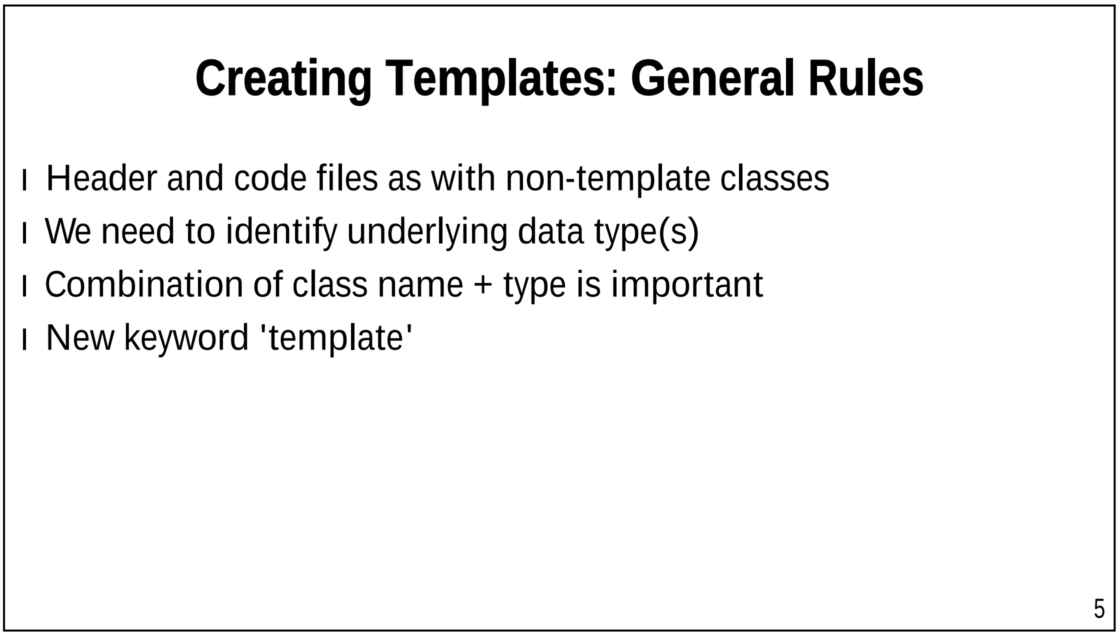
2

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

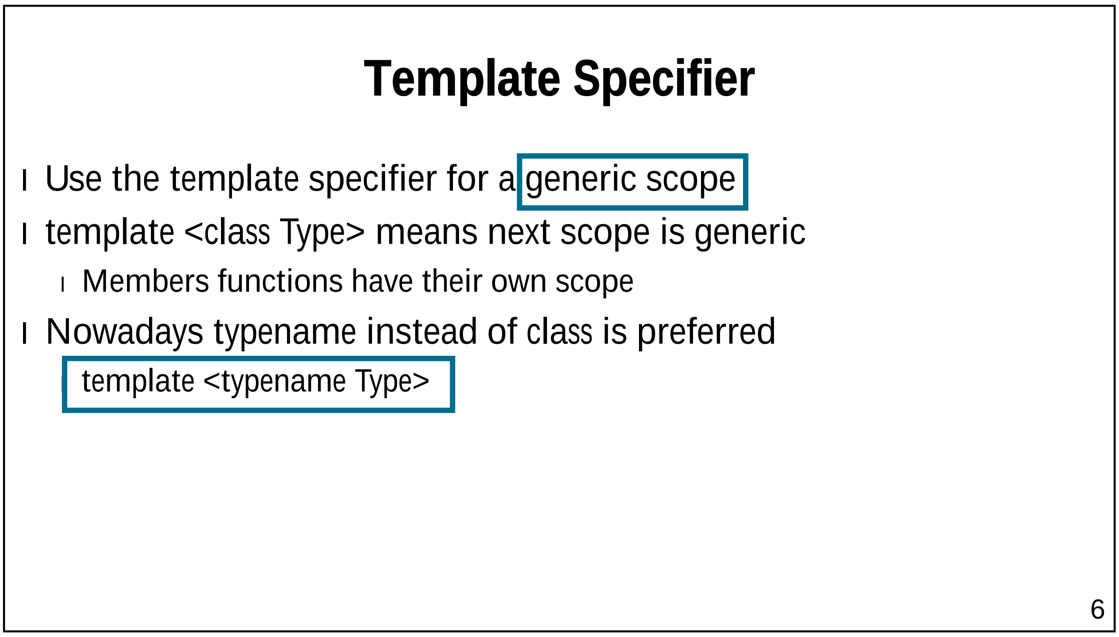
Templates

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



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'



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>
```

Example
Array State

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

private:
  Type* arr;
  int sz;

public:

};
```

When to Use Type

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

Type Usage

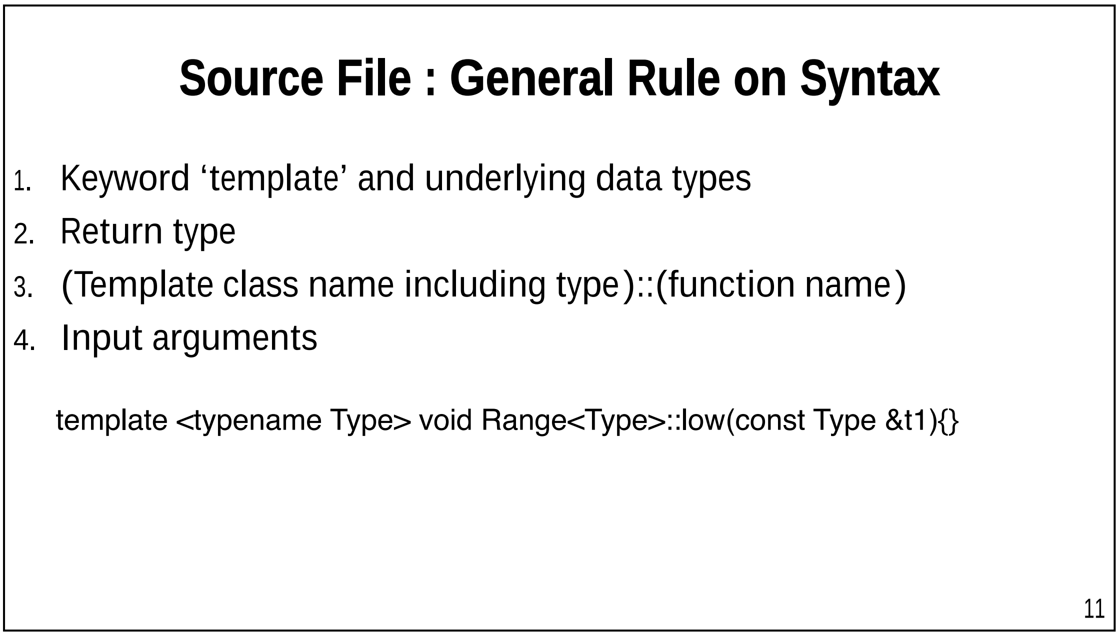
```
template <typename Type>
class Array
{
public:
  // Distinguishing between function names and class names

  // Function name
  Array();

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

Hints and Remarks

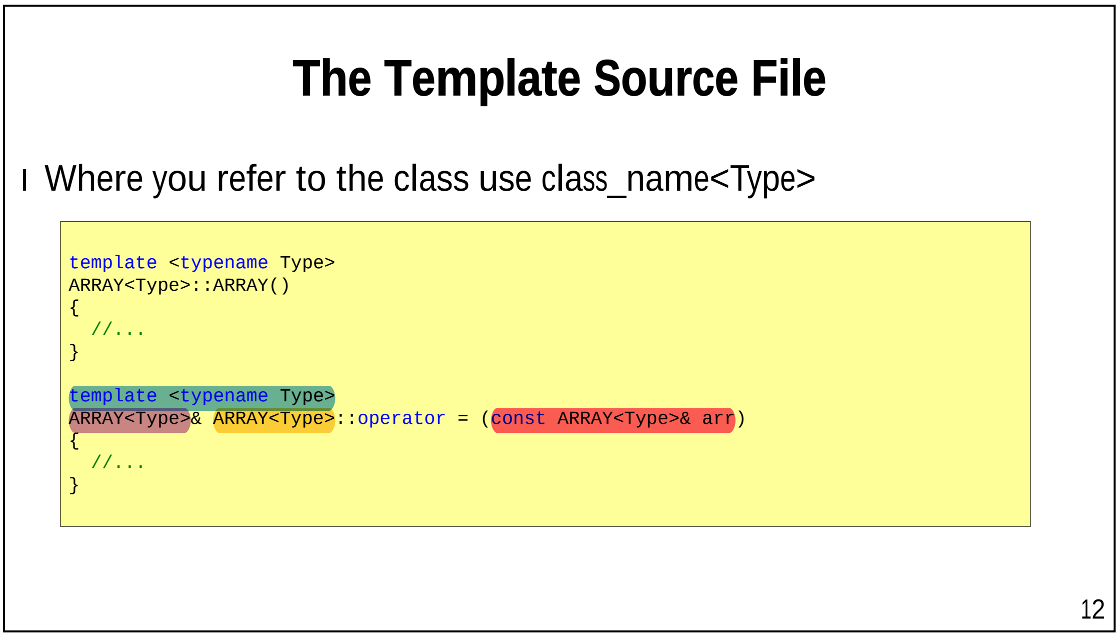
- Choose good names for underlying types
- Template class name == class name + type



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

template <typename Type> void Range<Type>::low(const Type &t1){}

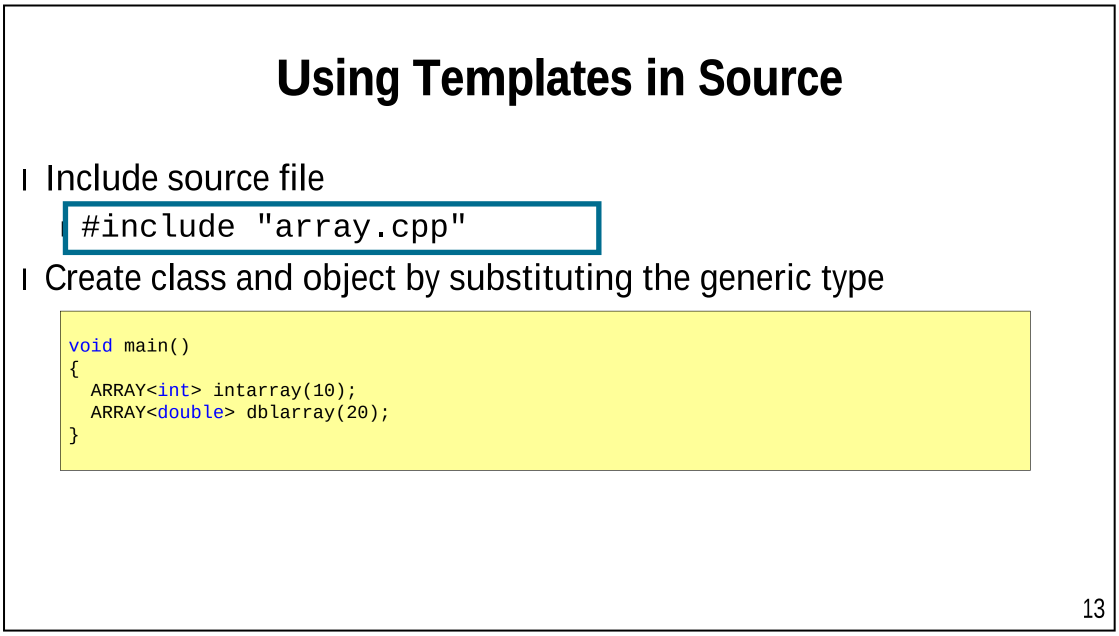


The Template Source File

- 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)
{
  //...
}
```



Using Templates in Source

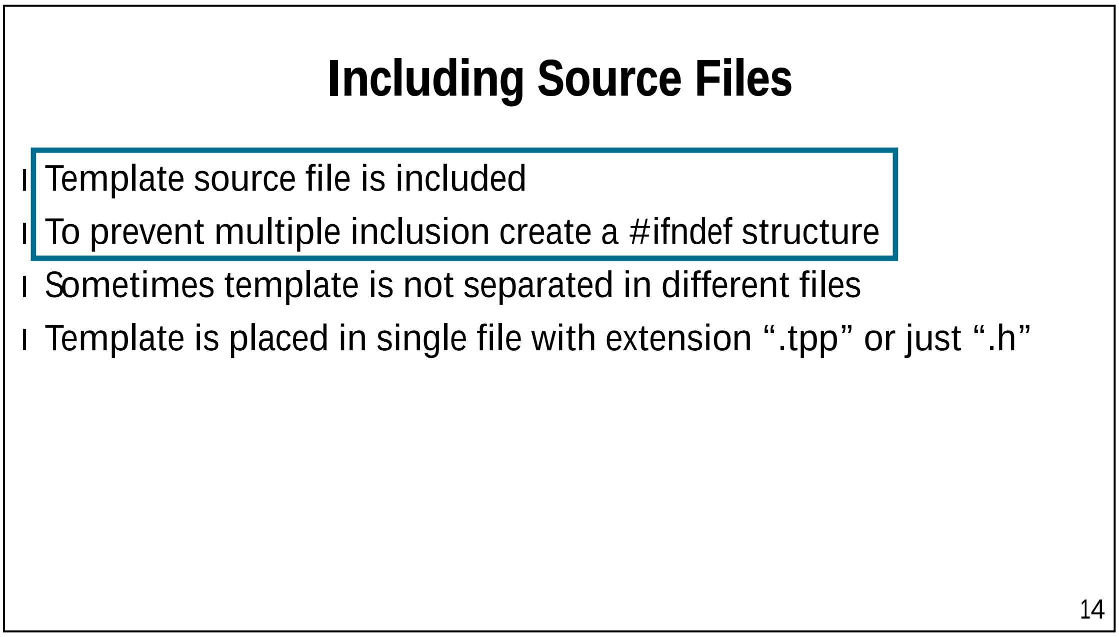
- Include source file
  - `#include "array.cpp"`
- Create class and object by substituting the generic type

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

13

Uninstanciated templates cannot be stored in an .o file for the linker. Thus, when using a template class, the implementation of the template class must be known at compile time while with regular classes the implementation of the class can be linked later at link time from an .o file. Three methods:

1. Implement the template class a inline in the header file;
2. Include the source file instead of header file;
3. The template class header file can include the template class source file if that was not yet done;



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

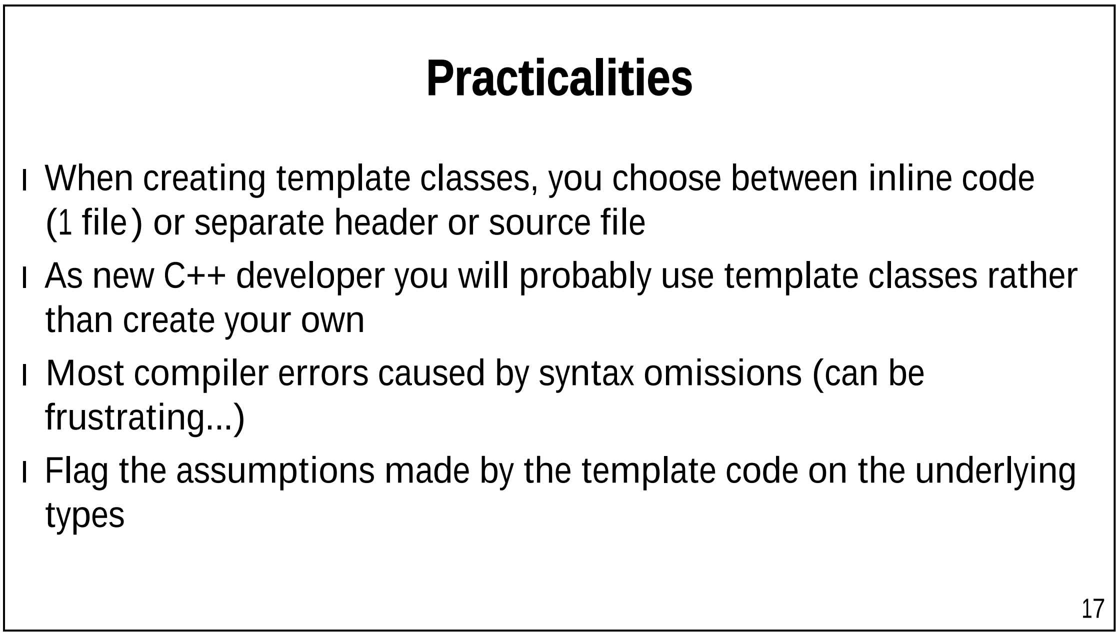
Generic Functions

- 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;
}
```

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



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