南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

C/C++ Program Design

CS205

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

http://faculty.sustech.edu.cn/yusq/

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Class Templates

Review: Function Templates

- A function template is not a type, or a function, or any other entity.
- No code is generated from a source file that contains only template definitions.
- The template arguments must be determined, then the compiler can generate an actual function
- "**Function templates**" vs "**template functions**".

```
template<typename T>
T sum(T x, T y)
{
   cout << "The input type is " << typeid(T).name() << endl;
   return x + y;
}

// instantiates sum<double>(double, double)
template double sum<double>(double, double);
// instantiates sum<char>(char, char), template argument deduced
template char sum<>(char, char);
// instantiates sum<int>(int, int), template argument deduced
template int sum(int, int);
```

Review: Function Templates

- Implicit instantiation occurs when a function template is not explicitly instantiated.

```
template<typename T>
T product(T x, T y)
{
    cout << "The input type is " << typeid(T).name() << endl;
    return x * y;
}

// Implicitly instantiates product<int>(int, int)
cout << "product = " << product<int>(2.2f, 3.0f) << endl;
// Implicitly instantiates product<float>(float, float)
cout << "product = " << product(2.2f, 3.0f) << endl;
```

Different Classes for Different Type Matrices

- Matrix with int elements, Matrix with float elements

```
class IntMat
{
    size_t rows;
    size_t cols;
    int * data;
  public:
    IntMat(size_t rows, size_t cols):
                    rows(rows), cols(cols)
    {
        data = new int[rows * cols * sizeof(int)]{};
    }
    ~IntMat()
    {
        delete [] data;
    }
    IntMat(const IntMat&) = delete;
    IntMat& operator=(const IntMat&) = delete;
    int getElement(size_t r, size_t c);
    bool setElement(size_t r, size_t
};
```

```
class FloatMat
{
    size_t rows;
    size_t cols;
    float * data;
  public:
    FloatMat(size_t rows, size_t cols):
                    rows(rows), cols(cols)
    {
        data = new float[rows * cols * sizeof(float)]{};
    }
    ~FloatMat()
    {
        delete [] data;
    }
    FloatMat(const FloatMat&) = delete;
    FloatMat& operator=(const FloatMat&) = delete;
    float getElement(size_t r, size_t c);
    bool setElement(size_t r, size_t c, float value);
```

matclass.cpp

Class Templates

- A class template defines a family of classes.
- Class template instantiation.

```
// Explicitly instantiate
template class Mat<int>;
```

```
template<typename T>
class Mat
{
   size_t rows;
   size_t cols;
   T * data;
 public:
   Mat(size_t rows, size_t cols): rows(rows), cols(cols)
   {
      data = new T[rows * cols * sizeof(T)]{};
   }
   ~Mat()
   {
      delete [] data;
   }
   T getElement(size_t r, size_t c);
   bool s                                    T value);
};
```

mattemplate.cpp

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Template Non-Type Parameters

Non-Type Parameters

- To declare a template

  template < parameter-list > declaration

- The parameters can be
  - type template parameters
  - template template parameters
  - non-type template parameters
    - ✓integral types
    - ✓floating-point type
    - ✓pointer types
    - ✓lvalue reference types
    - ✓…

```
vector<int> vec1;
vector<int, 16> vec2;
```

Non-Type Parameters

- If we want to create a static matrix (no dynamic memory allocation inside)

```
template<typename T, size_t rows, size_t cols>
class Mat
{
   T data[rows][cols];
 public:
   Mat(){}
   T getElement(size_t r, size_t c);
   bool setElement(size_t r, size_t c, T value);
};
```

```
Mat<int> vec1(3, 3);
Mat<int, 3, 3> vec2;
```


nontypeparam.cpp

Template in OpenCV

```
template<typename _Tp, int m, int n> class Matx
{
public:
    enum {
            rows     = m,
            cols     = n,
            channels = rows*cols,
#ifdef OPENCV_TRAITS_ENABLE_DEPRECATED
            depth    = traits::Type<_Tp>::value,
            type     = CV_MAKETYPE(depth, channels),
#endif
            shortdim = (m < n ? m : n)
          };
```

```
typedef Matx<float, 1, 2> Matx12f;
typedef Matx<double, 1, 2> Matx12d;
typedef Matx<float, 1, 3> Matx13f;
typedef Matx<double, 1, 3> Matx13d;
typedef Matx<float, 1, 4> Matx14f;
typedef Matx<double, 1, 4> Matx14d;
typedef Matx<float, 1, 6> Matx16f;
typedef Matx<double, 1, 6> Matx16d;
```

Template in OpenCV

```
template<typename _Tp, int cn> class Vec : public Matx<_Tp, cn, 1>
{
public:
    typedef _Tp value_type;
    enum {
        channels = cn,
#ifdef OPENCV_TRAITS_ENABLE_DEPRECATED
        depth    = Matx<_Tp, cn, 1>::depth,
        type     = CV_MAKETYPE(depth, channels),
#endif
        _dummy_enum_finalizer = 0
    };
```

```
typedef Vec<uchar, 2> Vec2b;
typedef Vec<uchar, 3> Vec3b;
typedef Vec<uchar, 4> Vec4b;

typedef Vec<short, 2> Vec2s;
typedef Vec<short, 3> Vec3s;
typedef Vec<short, 4> Vec4s;

typedef Vec<ushort, 2> Vec2w;
typedef Vec<ushort, 3> Vec3w;
typedef Vec<ushort, 4> Vec4w;

typedef Vec<int, 2> Vec2i;
typedef Vec<int, 3> Vec3i;
typedef Vec<int, 4> Vec4i;
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Class Template Specialization

Class template specialization

- The class template can be for most types
- But we want to save memory for type bool (1 byte or 1 bit).

```
template<typename T>
class MyVector
{
   size_t length;
   T * data;
 public:
   MyVector(size_t length): length(length)
   { data = new T[length * sizeof(T)]{}; }
   ~MyVector()
   { delete [] data; }
   MyVector(const MyVector&) = delete;
   MyVector& operator=(const MyVector&) = delete;
   T getElement(size_t index);
   bool setElement(size_t index, T value);
};
```

Class template specialization

- Specialize MyVector for bool

```
template<>
class MyVector<bool>
{
   size_t length;
   unsigned char * data;
 public:
   MyVector(size_t length): length(length)
   {
      int num_bytes = (length - 1) / 8 + 1;
      data = new unsigned char[num_bytes]{};
   }
   ~MyVector()
   {
      delete [] data;
   }
   MyVector(const MyVector&) = delete;
   MyVector& operator=(const MyVector&) = delete;
   bool getElement(size_t index);
   bool setElement(size_t index, bool value);
};
```

specialization.cpp

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

std classes

std::basic_string

- Store and manipulate sequences of char-like objects.

Defined in header <string>

Type	Definition
std::string	std::basic_string<char>
std::wstring	std::basic_string<wchar_t>
std::u8string (C++20)	std::basic_string<char8_t>
std::u16string (C++11)	std::basic_string<char16_t>
std::u32string (C++11)	std::basic_string<char32_t>
std::pmr::string (C++17)	std::pmr::basic_string<char>
std::pmr::wstring (C++17)	std::pmr::basic_string<wchar_t>
std::pmr::u8string (C++20)	std::pmr::basic_string<char8_t>
std::pmr::u16string (C++17)	std::pmr::basic_string<char16_t>
std::pmr::u32string (C++17)	std::pmr::basic_string<char32_t>

std::array

- a container that encapsulates fixed size arrays.

```
template<
    class T,
    std::size_t N
> struct array;
```

```
std::array<int, 3> a2 = {1, 2, 3};
```

*Keyword: typename/class, class/struct

Some other templates

```
template<
    class T,
    class Allocator = std::allocator<T>
> class vector;
```

```
template<
    class T,
    class Allocator = std::allocator<T>
> class list;
```

```
template<
    class Key,
    class Compare = std::less<Key>,
    class Allocator = std::allocator<Key>
> class set;
```

```
template<
    class Key,
    class T,
    class Compare = std::less<Key>,
    class Allocator = std::allocator<std::pair<const Key, T> >
> class map;
```

```
template<
    class T,
    class Container = std::deque<T>
> class stack;
```