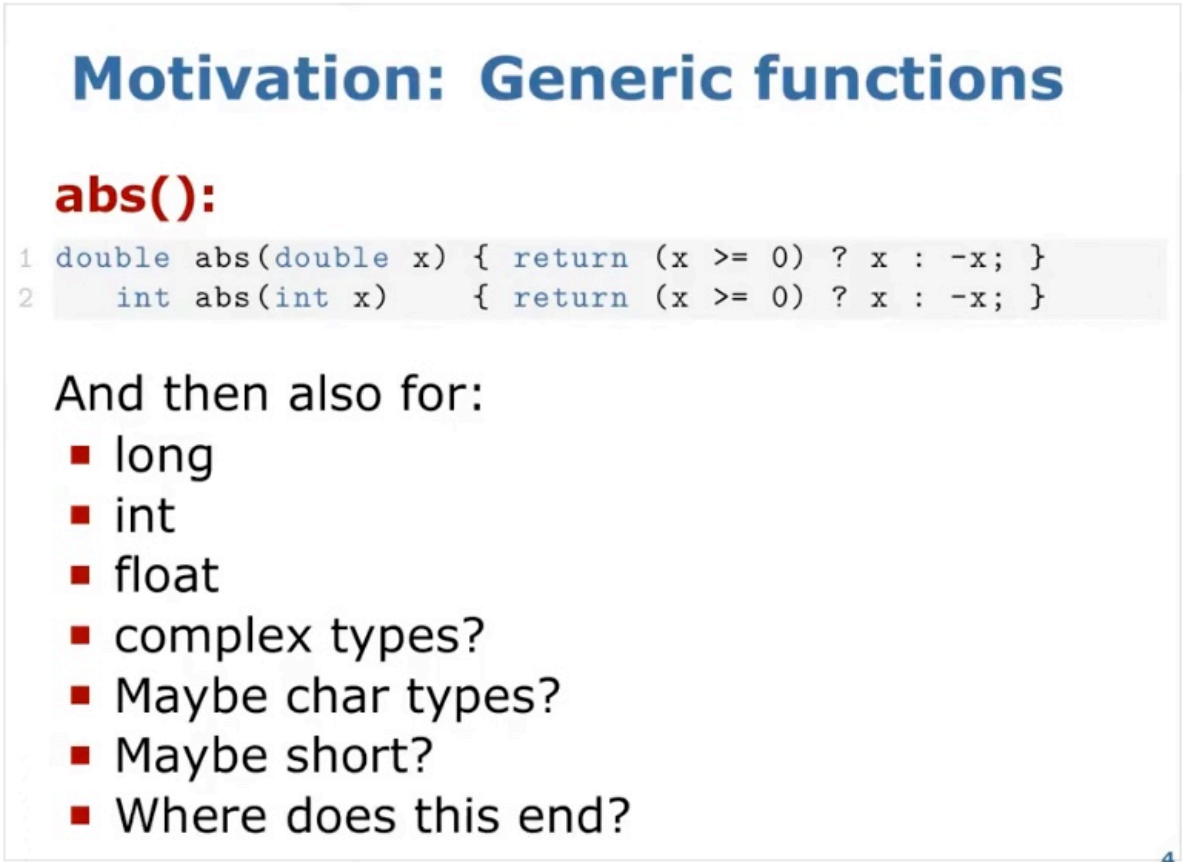
Templates

Meaning of templates

- Dictionary definitions:
  - Something that serves as a model for others to copy.
  - A preset format for a document or file.
  - Something that is used as a pattern for producing other similar things.
- C++ Definitions:
  A template is a C++ entity that defines one of the following:
  - A family of classes (class template), which may be nested classes.
  - A family of functions (function template), which may be member functions.

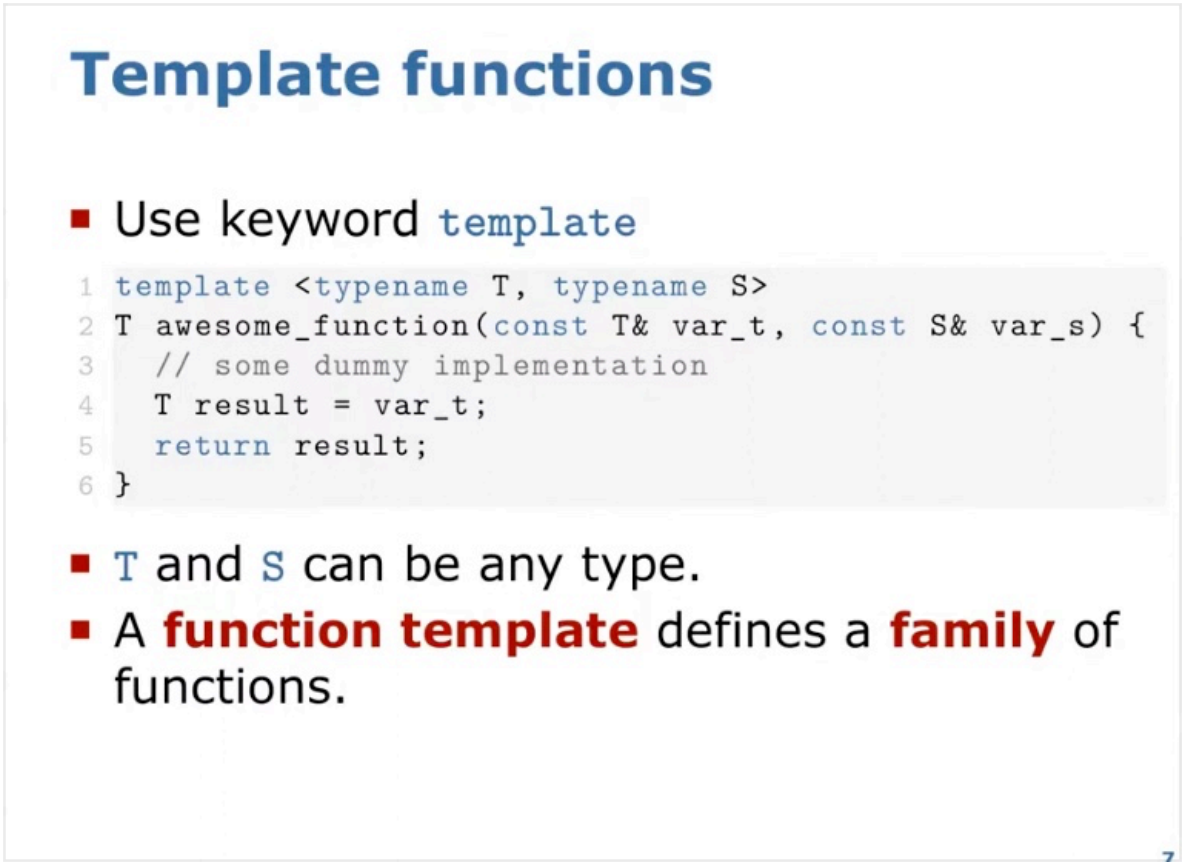


Function Templates

abs<T>():

```
template <typename T>
T abs(T x) {
  return (x >= 0) ? x : -x;
}
```

- Function templates are not functions. They are templates for making functions.
- Don’t pay for what you don’t use:
  If nobody calls abs<int>, it won’t be instantiated by the compiler at all.
- Templates live in a “static” world.



Using Function Templates

```
template <typename T>
T abs(T x) {
  return (x >= 0) ? x : -x;
}

int main() {
  const double x = 5.5;
  const    int y = -5;

  auto abs_x = abs<double>(x);
  int  abs_y = abs<int>(y);

  double abs_x_2 = abs(x);  // type-deduction
  auto   abs_y_2 = abs(y);  // type-deduction
}
```

8

Template Classes

```
template <class T>
class MyClass {
 public:
  MyClass(T x) : x_(x) {}

 private:
  T x_;
};
```

- Classes templates are not classes. They are templates for making classes.
- Don’t pay for what you don’t use:
  If nobody calls MyClass<int>, it won’t be instantiated by the compiler at all.

Template classes usage

```
template <class T>
class MyClass {
 public:
  MyClass(T x) : x_(x) {}

 private:
  T x_;
};

int main() {
  MyClass<int> my_float_object(10);
  MyClass<double> my_double_object(10.0);
  return 0;
}
```

10

Template Parameters

- Every template is parameterised by one or more template parameters: template <parameter-list> declaration;

```
template <typename T, size_t N = 10>
T AccumulateVector(const T& val) {
  std::vector<T> vec(val, N);
  return std::accumulate(vec.begin(), vec.end(), 0);
}
```

- Think the template parameters the same way as any function arguments, but at compile time.

Template Parameters

```
1  template <typename T, size_t N = 10>
2  T AccumulateVector(const T& val) {
3    std::vector<T> vec(val, N);
4    return std::accumulate(vec.begin(), vec.end(), 0);
5  }
6
7  using namespace std;
8  int main() {
9    cout << AccumulateVector(1) << endl;
10   cout << AccumulateVector<float>(2) << endl;
11   cout << AccumulateVector<float, 5>(2.0) << endl;
12   return 0;
13 }
```