

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.

Motivation: Generic functions

abs():

```
1 double abs(double x) { return (x >= 0) ? x : -x; }
2 int abs(int x)      { return (x >= 0) ? x : -x; }
```

And then also for:

- long
- int
- float
- complex types?
- Maybe char types?
- Maybe short?
- Where does this end?

Function Templates

abs<T>():

```
1 template <typename T>
2 T abs(T x) {
3     return (x >= 0) ? x : -x;
4 }
```

- 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.

Template functions

- Use keyword `template`

```
1 template <typename T, typename S>
2 T awesome_function(const T& var_t, const S& var_s) {
3     // some dummy implementation
4     T result = var_t;
5     return result;
6 }
```

- `T` and `S` can be any type.
- A **function template** defines a **family** of functions.

Using Function Templates

```
1 template <typename T>
2 T abs(T x) {
3     return (x >= 0) ? x : -x;
4 }
5
6 int main() {
7     const double x = 5.5;
8     const     int y = -5;
9
10    auto abs_x = abs<double>(x);
11    int  abs_y = abs<int>(y);
12
13    double abs_x_2 = abs(x); // type-deduction
14    auto   abs_y_2 = abs(y); // type-deduction
15 }
```

R

Template Classes

```
1 template <class T>
2 class MyClass {
3     public:
4         MyClass(T x) : x_(x) {}
5
6     private:
7         T x_;
8 };
```

- 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

```
1 template <class T>
2 class MyClass {
3     public:
4         MyClass(T x) : x_(x) {}
5
6     private:
7         T x_;
8 };
9
10 int main() {
11     MyClass<int> my_float_object(10);
12     MyClass<double> my_double_object(10.0);
13     return 0;
14 }
```

10

Template Parameters

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

```
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 }
```

- 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 }
```