ADVANCED PROGRAMMING IN C++

Generic Programming

Patrick Bader · SS 2023

Abstractions

Previously discussed abstractions are based on types

Generic Programming

Write code that does not depend on concrete types

Function Template Example

Declaration and definition:

```
template<typename T>
T max(const T& a, const T&b) {
  return a > b ? a : b;
}
```

Usage:

```
float f = max(1.0f, 2.0f);     // implicitly call max<float>
int i = max(3, 6);             // implicitly call max<int>
double d = max<double>(i, f); // explicitly call max<double>
```

Class Template Example

Declaration and definition:

```
template<typename T, int Columns, int Rows >
class Matrix {
  // Template parameters are known at compile time, so can be used in constant expressions
  // e.g. to calculate size of a C-array
  T m_data[Columns * Rows];
public:
  const T& operator()(int column, int row) const {
    return m_data[row * Columns + column];
  }
};
```

Usage:

```
Matrix<float, 3, 4> m ;
float f = m(1, 2);
```

Templates - Summary

Allow functions and classes to be **generalized to arbitrary types** and values.

Instantiated at **compile-time** to concrete classes.

One instance **for each Parameter combination** in use.

Can be instantiated with **every type** that allows the **used operations**.

Templates - Summary

Definition must be visible to compiler when template is instantiated

Parameters can be:

- Type-parameters
- Nontype-parameters
- Templates
- Parameter packs