Introduction

This is a structured list of the contents of the lecture "Object-Oriented Programming for Scientific Computing", as held by Ole Klein in summer 2017 at Heidelberg University.

This list serves three purposes, it is:

- a guide for exam preparations
- a list of suggestions for further self-guided studies
- a working draft for upcoming lectures

Almost all topics discussed in the lecture are included in the following list, roughly in order of appearance. Topics that have their own detailed list are marked by "→". Topics that have not been included (either on purpose or due to time restrictions) are marked like [*this*].

The inclusion of a topic does not signify its importance, nor does its omission mean that it is less significant. Suggestions for improvement are welcome at `ole.klein@iwr.uni-heidelberg.de`.

The C++ Language

Specific core concepts, programming paradigms and standard libraries that characterize the C++ language.

- programming paradigms → 0.1
- central concepts → 0.2
- standard libraries → 0.3
- memory management → 6.1
- smart pointers (C++11) → 6.4
- exceptions → 3.3
- UML representation → 3.5

0.1 Programming Paradigms

- procedural programming → 1.1
- object-based programming → 1.2
- object-oriented programming → 1.3
- generic programming → 1.4
- compile-time programming → 1.5

0.2 Central Concepts

- types → 2.5
- namespaces → 3.2
- constness → 6.3
- RAII → 3.4

0.3 Standard Libraries

- container library (STL) → 7
- [*input/output library*]
- [*thread support library (C++11)*]
- [*random number generators (C++11)*]
- chrono library (C++11)
- regular expressions library (C++11)
- [*filesystem library (C++17)*]
- [*mathematical function library (C++17)*]

1 Programming Paradigms

Different programming paradigms supported by C++, from procedural programming to compile-time programming techniques.

1.1 Procedural Programming

- variables → 2.1
- references → 2.2
- pointers → 2.3
- functions → 2.4
- built-in types → 2.6
- control structures → 3.1
- move semantics (C++11) → 6.2

1.2 Object-Based Programming

- classes → 4.1
- objects
- encapsulation → 4.2
- inheritance → 4.4

1.3 Object-Oriented Programming

- dynamic polymorphism
- virtual methods
  - `override` (C++11)
  - vtable
  - vpointer
- abstract base class
  - pure virtual
  - interface base class
- arrays of derived objects
  - virtual destructors
- `dynamic_cast`
- "virtual constructors"
- virtual base classes

1.4 Generic Programming

- templates → 5.1
- static polymorphism
- [*Lambdas with* `auto` *(C++14)*]

1.5 Compile-Time Programming

- template metaprogramming
  - `is_same`
  - `enable_if`
  - `conditional`
  - [*`static_assert`*]
  - [*SFINAE*]
- `constexpr` (C++11)
  - [*`if constexpr` (C++17)*]

2 Variables and Functions

Central constructs of imperative / procedural programming, building blocks of the C++ language.

2.1 Variables

- scope / lifetime
- built-in types → 2.6
- automatic type deduction
  - `auto` (C++11)
  - `decltype` (C++11)
  - [*`template <auto>` (C++17)*]
- initialization
  - default initialization
  - value initialization
  - [*list initialization (C++11)*]
  - structured bindings (C++17)
- objects

2.2 References

- const
- dangling references
- rvalue references
- lifetime extension

2.3 Pointers

- dereferencing
- increment / decrement
- C arrays
- dangling pointers
- segmentation fault
- `nullptr` (C++11)

2.4 Functions

- arguments
- default arguments
- return type
- pass-by-value
- pass-by-reference
- function overloading
- operators
- [*lambda functions (C++11)*]

2.5 Types

- built-in types → 2.6
- classes → 4.1
- `typedef`
- `using` alias (C++11)
- [*strings*]
- [*streams*]
- pair
- tuple (C++11)
- [*valarray*]
- [*bitset*]

2.6 Built-in Types

- `bool`
- `char`
- `short`
- `int`
- `long`
- `long long` (C++11)
- `float`
- `double`
- `long double`

3 Organization and Control Flow

Constructs for the organization of code, specification of program flow and general structuring.

3.1 Control Structures

- if / else
  - [*`if constexpr` (C++17)*]
  - if with initializer (C++17)
- loops
  - range-based for (C++11)
- case / switch
  - [*switch with initializer (C++17)*]

3.2 Namespaces

- keyword `using`
- nested classes

3.3 Exceptions

- `throw`
- `try` block
  - [*function* `try` *block*]
- `catch` block
- `noexcept` (C++11)
  - `noexcept` specifiers (C++11)
  - `noexcept` operator (C++11)
- inheritance
- `exception_ptr` (C++11)
- `assert`

3.4 RAII

- destruction is resource release (DIRR)
- exception safety
- recursive destructor calls

3.5 UML Representation

- class diagram
  - attributes
  - is-a
  - has-a
  - uses-a
  - knows-a
  - interfaces
  - templates

4 Classes and Objects

Definition and implementation of classes, inheritance and class hierarchies.

4.1 Classes

- declaration
- definition
  - inline
  - after declaration
- attributes
  - `const`
  - `mutable`
  - `static`
- methods
  - `const`
  - `static`
  - constructor → 4.3
  - destructor
  - assignment operator
- `struct`
- `this` pointer

4.2 Encapsulation

- interface
  - `public`
- implementation
  - `private`
  - `protected`
- `friend`

4.3 Constructor

- default constructor
- copy constructor
- move constructor
- initialization list
- default member values (C++11)
- delegating constructor (C++11)

4.4 Inheritance

- base class
- derived class
- class invariant
- `public`
  - scoping
  - is-a
  - slicing
- `protected`
- `private`
  - has-a
- covariant return types
- multiple inheritance
  - exceptions
  - multiple interfaces
  - deadly diamond of death
- inheriting constructors (C++11)
- `final` (C++11)

5 Templates

Templates as a tool for generic programming, policies and traits classes.

5.1 Templates in General

- function templates
  - specialization
- class templates → 5.2
- variable templates (C++14)
- instantiation
- template parameters
  - type template parameters
  - non-type template parameters
  - template template parameters
  - template default arguments
- `using` alias (C++11)
- keyword `.template`
- variadic templates (C++11)
  - template parameter pack (C++11)
  - expansion loci (...) (C++11)
  - fold expression (C++17)

5.2 Class Templates

- specialization
- partial specialization
- class name shorthand
- member templates
- dependent types
  - keyword `.typename`
- templatized inheritance
  - independent base classes
  - dependent base classes
  - delayed type resolution

- traits
  - type traits
  - value traits
  - promotion traits
- policies
- [*class template deduction (C++17)*]

6 Memory

Memory layout and constructs for the efficient and safe use of memory.

6.1 Memory Management

- static memory
  - global variables
  - static variables
- stack (automatic)
  - local variables
  - temporaries
- heap (dynamic)
  - new / delete
  - pointers
  - memory leak
  - new
  - / delete
  -
- [*thread_local (C++11)*]
- [*code area*]
- alignment
  - sizeof operator
  - [*alignof operator (C++11)*]

6.2 Move Semantics (C++11)

- rvalue reference
- move constructor
- move-assignment operator

6.3 Constness

- variables
- references
- pointers
- methods
  - `const` overload
- attributes
  - `mutable`
- [*`volatile`*]

6.4 Smart Pointers (C++11)

- `unique_ptr` (C++11)
  - `make_unique` (C++14)
- `shared_ptr` (C++11)
  - manager object
  - shared counter
  - `shared_from_this`
  - `make_shared`
- `weak_ptr` (C++11)
  - weak counter

7 C++ Container Library (STL)

Four sublibraries (container, iterator, algorithm, functor) of the standard library with central importance.

- STL containers → 7.1
- STL iterators → 7.2
- STL algorithms → 7.3
- STL functors

7.1 STL Containers

- sequences
  - array (C++11)
  - deque
  - forward_list (C++11)
  - list
  - vector
- sequence adaptors
  - stack
  - queue
  - priority_queue
- sorted associative containers
  - set
  - multiset
  - map
  - multimap
- [*unordered associative containers (C++11)*]
  - [*unordered_set (C++11)*]
  - [*unordered_multiset (C++11)*]
  - [*unordered_map (C++11)*]
  - [*unordered_multimap (C++11)*]

7.2 STL Iterators

- OutputIterator
- ForwardIterator
- BidirectionalIterator
- RandomAccessIterator
- const_iterator

7.3 STL Algorithms

- iterator ranges
- information
  - counting
  - extrema
  - search
  - comparison
- manipulation
  - copy / move
  - transform
  - remove
- reordering
  - shuffle (C++11)
  - partition
  - heap
  - sort
- for sorted ranges
  - binary search
  - set operations
- numerical algorithms
- execution policies (C++17)
  - parallel numerical algorithms (C++17)