Software Engineering

developed since January 2018

Course Modules

01 General Introduction

Program: 2 or 3 lectures: duration up to 6 hours in total; 1 programming problem for homework.

Abstract: Definition. Standards. Applications. Programming paradigms. Libraries. Tools. Preparations. Git.

Definition. Evolution. Standards. Applications. Programming paradigms. Declarative, functional, imperative, procedural, structured, object-oriented, generic, parallel, event-driven programming. Language core. Standard template library. STL. Boost. Additional libraries. Developer instruments. Integrated development environments. Code editors. Compilers. Debuggers. Profilers. Build automation systems. Version control systems. Project management systems. Important persons and external resources. Operating systems. Windows. Visual Studio. MSVC. Solutions and projects. Source files. Configurations. Debug and release. x86 and x64. Build. Errors and warnings. Linux. Ubuntu. Terminal. GCC. Visual Studio Code. Extensions. Properties. Coding styles. Formatting. Naming. `snake_case`, `camelCase`. Hungarian notation. STL, Boost and Google coding styles. Comments. Documentation. Doxygen. Hello, world! Main function. Console. Git. GitHub. SmartGit. Repositories. Commands. Clone, commit, push, pull. Discard, revert, reset. Branches. Master. Merge and rebase. Logs. Conflicts. Index. Continuous integration. CI. Continuous deployment. CD.

Educational Materials

- `01.01.introduction`
- `01.02.introduction.function.main`
- `01.03.introduction.standard`
- `01.04.project.tool.git`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 1.2, 1.4, 44
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 1.2, 8, 16
- B. Stroustrup, The Design and Evolution of C++, sections 1-9
- N. Josuttis, The C++ Standard Library, 2nd Edition, section 2.1
- S. Meyers, Effective C++, 3rd Edition, sections 1, 53-55
- S. Meyers, More Effective C++, section 35
- S. Dewhurst, C++ Gotchas, sections 1, 11, 12
- H. Sutter, C++ Coding Standards, sections 0-4
- R. Grimm, C++20 Get the Details, sections 1, 2

02 Programming Basics

Program: 2 or 3 lectures: duration up to 8 hours in total; 13 programming problems for homework.

Abstract: Variables. Types. Conversions. Expressions. Statements. Pointers. Arrays. References. Functions.

Declarations and definitions. One definition rule. ODR. Objects. Variables. Types. Fundamental types. Arithmetic types. Integral types. `bool`, `char`, `short`, `int`, `long`, `float` , `double`. `long long`, `long double`. Size in bytes. `sizeof`. Binary format for representing numbers. Overflow problem. Single and double precisions. Two's complement. `signed`, `unsigned`. Portability problem. Implementation-dependent types. Literals. Boolean, character, integral and floating-point literals. `true`, `false`. Literal suffixes. Values. Initialization. Undefined behavior. Default, value, copy, list, uniform initialization. Type deduction. `auto`. Conversions. Temporary objects. Implicit and explicit conversions. Narrow and wide conversions. Integral promotions. Old style conversions. `static_cast`. Magic numbers. Constants. `const`. Special memory. `volatile`. Type aliases. `using`, `typedef`. `std::size_t`. Attributes. `[[maybe_unused]]`.

Operators. Arity. Nullary, unary, binary, ternary operators. Precedence. Parentheses. Logical operators. Alternative representations. `or`, `and`, `not`. Short-circuit evaluations. Arithmetic operators. Arithmetic operators with assignment. Comparison operators. Prefix and postfix increment and decrement operators. Operands evaluation order. Side effects. Unspecified behavior. Sequences of operators. Associativity. Assignment operator. Comma operator. Expressions. Control flow. Conditions. Selections. `if`, `else`. Ternary operator. `switch`, `case`, `default`. `[[fallthrough]]`. Optimization. `[[likely]]`, `[[unlikely]]`. Loops. `for`, `while`, `do`. Jumps. `continue`, `break`, `goto`. Labels. Statements.

Memory addressing. Pointers. Address-take and dereference operators. Null pointers. `nullptr`. `std::nullptr_t`. Constant pointers. Pointers to constants. Arrays. Static arrays. Aggregate initialization. Indexing. Size of arrays. `std::size`. Pointer arithmetic. Arrays and pointers. `std::swap`. Dynamic arrays. `new`, `new[]`, `delete`, `delete[]`. Containers. `std::vector`. Range-based for loop. Lvalue references. Constant references. `std::reference_wrapper`.

Functions. Function scopes. Forward declarations. Definitions and declarations. Function bodies. Pre- and postconditions. Parameters and arguments. Arguments evaluation order. Default arguments. Passing arguments by values, pointers and references. Sequences and views. `std::span`. Static and dynamic extents. Returning results. `return`. `[[nodiscard]]`. `void`. Trailing return types. `auto`. Inline functions. `inline`. Recursion. Returning dangling pointers and references. Local variables. Static variables. `static`. Functions overloading. Function signatures. Overloading resolution. Binary search. `std::midpoint`. Lexicographic order. Hybrid insertion-merge sorting. Amortized complexity.

Educational Materials

- `02.01.statement.expression.variable.type`
- `02.02.statement.expression.operator`
- `02.03.statement.selection`
- `02.04.statement.iteration.jump`
- `02.05.pointer`
- `02.06.pointer.array`
- `02.07.pointer.array.new.delete`
- `02.08.container.vector`
- `02.09.reference.lvalue`
- `02.10.function`
- `02.11.algorithm.binary.search`
- `02.12.algorithm.lexicographic.order`
- `02.13.algorithm.insertion.merge.sorting`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 6, 7, 9, 10.3-10.5, 11.1, 11.2, 11.5, 12.1-12.4
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 1, 3.6, 13.3
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 2.2, 5.5
- S. Meyers, Effective C++, 3rd Edition, sections 16, 20, 21, 26, 27, 30
- S. Meyers, More Effective C++, sections 1, 2, 16, 23
- S. Meyers, Effective Modern C++, sections 2.1, 3.1-3.3, 8.1
- S. Dewhurst, C++ Gotchas, sections 2, 4-7, 9, 13-18, 20, 21, 31, 32, 40, 41, 44, 48, 60, 66
- H. Sutter, C++ Coding Standards, sections 5-9, 15, 17-20, 25, 31, 77, 91, 95, 99
- R. Grimm, C++20 Get the Details, sections 4.8, 5.2, 5.4
- M. Bancila, The Modern C++ Challenge, section 1

03 Classes Development

Program: 4 or 5 lectures: duration up to 16 hours in total; 6 programming problems for homework.

Abstract: Object-oriented programming. Structs. Classes. Members. Copy and move semantics. Operators.

Object-oriented programming. User-defined types. Structures. `struct`. Data members. Plain old data. Instances. Default initialization. Aggregate data types. Aggregate initialization. Designated initialization. Arrays of structures. Members selection. Dot-arrow operators. Constant structures. Passing structures by reference. Returning structures.

Classes. `class`. Class scopes. Encapsulation. Access specifiers. `public`, `private`. Interface and implementation. Member functions. Invariants. Inline and external member function definitions. Constant and non-constant member functions. `const`. Bitwise and logical constancy. `mutable`. Data cache. Getters and setters. Constructors. Default constructors. Implicit default constructors. User-defined main constructors. Delegating constructors. Initializer lists. Members initialization order. Destructors. Nested aliases and types. Static members. `static`. Variables, constants and functions. Class friends. `friend`. Restricting access to private members. Attorney-client pattern. Passkey idiom.

Types of relationships. Composition, aggregation, association and dependency. Inheritance. Class hierarchies. Base and derived classes. Constructor and destructor procedures. Protected members. `protected`. Public, protected and private inheritance. Interface inheritance. Is a variety of. Implementation inheritance. Implemented through. Calling inherited functions. Changing access specifiers. Multiple inheritance. Diamond problem solution. Virtual inheritance. Polymoprhism. Onion object. Objects slicing. Upcasting. Virtual functions. `virtual`. Additional specifiers. `override`, `final`. Virtual destructors. Early and late bindings. Virtual tables. Virtual pointers. Covariant return types. Abstraction. Pure virtual functions. Abstract base classes. Interface classes. Empty classes. Sharing same addresses. `[[no_unique_address]]`. Microsoft Visual C++ issues. Compressed pair. Empty base class optimization. Runtime types identification. Downcasting. `dynamic_cast`. Runtime checks. Types as types. `decltype`, `decltype(auto)`. Types as strings. `typeid`. Problem with references. Boost.TypeIndex. `std::any`, `std::make_any`, `std::any_cast`.

Copy and move semantics. Equivalence and independence properties. Expression categories. History. Lvalues. Rvalues. Possibility of taking an address. Identifiability. Moveability. Rvalues and temporary objects. Prvalues. Generalized lvalues. Xvalues. Rvalue references. Overload resolution. Reference qualifiers. Special member functions. Copy and move constructors and assignment operators. Deep and shallow copying. Self-assignments. Hidden pointers. `this`. Member function chains. Swap functions. Copy and swap. CaS idiom. Making objects movable. `std::move`. Object states after moving. Special member functions generating rules. Rules of 0, 3, 4 and 5. Default implementations. Optimizations by compiler. Copy elision. Return value optimization. RVO. Named return value optimization. NRVO.

Operators overloading. Explicit and implicit conversions. Explicit and implicit constructors and typecasts. `explicit`. Overloading operators as member, free and friend functions. Overloading input and output operators. Overloading prefix and postfix increment and decrement operators. Overloading comparison operators. Operator-like and function-like calling styles. Rational arithmetic. Boost.Rational. Output operators in hierarchies. Three-way comparison. Spaceship operator. Rewriting expressions. Strong, weak and partial orderings. `std::strong_ordering`, `std::weak_ordering`, `std::partial_ordering`. Lexicographical comparisons. Operators optimization. Removing constancy. `const_cast`.

Educational Materials

- `03.01.class.struct`
- `03.02.class`
- `03.03.class.friend`
- `03.04.pattern.attorney.client.passkey`
- `03.05.class.association`
- `03.06.class.inheritance`
- `03.07.class.inheritance.multiple`
- `03.08.class.inheritance.multiple.diamond`
- `03.09.class.polymorphism`
- `03.10.class.polymorphism.vptr.vtbl`
- `03.11.class.polymorphism.type.covariant`
- `03.12.class.polymorphism.type.identification`
- `03.13.class.optimization`
- `03.14.expression.classification`
- `03.15.reference.lvalue.rvalue`
- `03.16.reference.lvalue.rvalue.function.overload`
- `03.17.implementation.container.vector`
- `03.18.class.operator.overload`
- `03.19.class.operator.overload.output`
- `03.20.class.operator.overload.spaceship`
- `03.21.class.operator.overload.optimization`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 6.4, 7.7, 8.2, 13.6, 16-22
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 2.2, 2.3, 4, 5
- D. Vandevoorde, C++ Templates, 2nd Edition, sections 21.1, B, C
- S. Meyers, Effective C++, 3rd Edition, sections 3-5, 7, 9-12, 18, 19, 21-25, 28, 32-40
- S. Meyers, More Effective C++, sections 2-7, 17-22, 24, 31, 33
- S. Meyers, Effective Modern C++, sections 1.3, 1.4, 3.6, 3.11, 5
- S. Dewhurst, C++ Gotchas, sections 10, 19, 22, 23, 30, 33, 35-39, 42, 45-47, 49-54, 56-59, 69-71, 73-87, 89-99
- H. Sutter, C++ Coding Standards, sections 11, 26-30, 32-38, 40-42, 44, 47-50, 52, 54-56, 90, 93, 94, 100
- R. Grimm, C++20 Get the Details, sections 4.4, 4.8
- M. Bancila, The Modern C++ Challenge, sections 2.15, 2.16

04 Types and Templates

Program: 4 or 5 lectures: duration up to 16 hours in total; 11 programming problems for homework.

Abstract: Generic programming. Templates. Specializations. Metaprogramming. Traits. Concepts. Tuples.

Generic programming. Templates. `template`. Two-phase templates translation. Ordinary lookup rules. Points of instantiation. Inclusion model. Separation model. Explicit instantiation options. Template parameters and arguments. Function templates. Type template parameters. `typename`. Default template arguments. Abbreviated function templates. Function template specializations. Full specializations. Overloading function templates. Non-type template parameters. Templates for built-in arrays. Variadic templates. Template parameters packs. Ellipsis. Recursive templates instantiation. Unpacking arguments. Fold expressions. Variadic traverse. Pointers to class members. Operator arrow-dereference. Variadic expressions. Class templates. Member functions instantiation rules. Templated entities. Template template parameters. Nested templates. Class template arguments deduction. Deduction guides. Class template specializations. Full and partial specializations. Class template friends. Special templates. Alias and variable templates.

Templates metaprogramming. Turing-complete languages. Generative programming. Policy-based programs. Source optimization. As-if rule. Compile-time calculations. Constant expressions. Immediate functions. `constexpr`, `consteval`, `constinit`. Compile-time selections. `if constexpr`. Hybrid metaprogramming. Compile-time ratios and durations.

Fundamental properties. Modifiability, constancy and moveability. Lvalue and rvalue references. Templates with move semantics. Perfect forwarding. Forwarding references. `std::forward`. Generic invocation. `std::invoke`. Special member function templates. Substitution failure is not an error. SFINAE idiom. `std::enable_if`. Passing arguments by value and by lvalue, rvalue, forwarding references. Types inference. Decay conversions. Reference convolution rules.

Type traits. Compile-time template-based interfaces. Implementation details. Class templates. Full and partial specializations. Standard base classes. `std::integral_constant`, `std::false_type`, `std::true_type`. Static constants and aliases. Alias and variable templates. Type properties. Type relationships. Type transformations. Conditional type traits. Unevaluated contexts. `std::declval`, `decltype`. Policies. `static_assert`. Concepts. `concept`. Constraints. Constrained type template parameters. Constrained placeholder types. Compile-time predicates. Conjunctions and disjunctions. Requirements. Requires expressions. Requires clauses. `requires`. Named requirements. Standard concepts.

Variadic heterogeneous lists of types. Variadic heterogeneous collections. `std::pair`. Tuples. `std::tuple`. Making tuples. `std::make_tuple`. Helper classes. `std::tuple_size`, `std::tuple_element`. Accessing elements. Index-based and type-based accesses. `std::get`. Unpacking tuples. `std::tie`. Placeholders. `std::ignore`. Ignoring results of nodiscard functions. Concatenating tuples. `std::tuple_cat`. Structured bindings. Recursive tuple printer helper.

Educational Materials

- `04.01.template.function`
- `04.02.template.function.variadic`
- `04.03.template.function.variadic.tree.traversal`
- `04.04.template.class`
- `04.05.template.class.friend`
- `04.06.template.alias.variable`
- `04.07.template.metaprogramming`
- `04.08.template.metaprogramming.constexpr`
- `04.09.template.metaprogramming.constexpr.hybrid`
- `04.10.reference.lvalue.rvalue.forwarding`
- `04.11.pattern.sfinae`
- `04.12.template.type.inference`
- `04.13.implementation.template.type.trait`
- `04.14.implementation.template.type.concept`
- `04.15.implementation.utility.tlist`
- `04.16.implementation.utility.tuple`
- `04.17.utility.tuple`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 10.4, 24.2-24.4, 25, 26, 28.2-28.7, 35.3-35.5
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 6, 7.2-7.5, 13.4, 13.9
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 5.1, 5.4, 5.6
- D. Vandevoorde, C++ Templates, 2nd Edition, sections 1-17, 19, 20, 23-25, 27, 28, D, E
- S. Meyers, Effective C++, 3rd Edition, sections 41-46, 48
- S. Meyers, Effective Modern C++, sections 1.1, 1.2, 3.3, 3.9, 5
- S. Dewhurst, C++ Gotchas, section 88
- H. Sutter, C++ Coding Standards, sections 65-67
- R. Grimm, C++20 Get the Details, section 4.1, 4.5, 4.6
- M. Bancila, The Modern C++ Challenge, sections 2.18, 2.19

05 Patterns and Idioms

Program: 2 or 3 lectures: duration up to 8 hours in total; 19 programming problems for homework.

Abstract: Generative, structural and behavioral design patterns. Dynamic and static polymorphism. Mixins.

Object-oriented programming. SOLID principles. Single responsibility principle. Open-closed principle. Liskov substitution principle. Interface segregation principle. Dependency inversion principle. Gang of Four design patterns and idioms. Generative patterns. Builder. Factory. Factory method. Abstract factory. Prototype. Virtual construction as cloning. Antipatterns. Singleton. Default and deleted special member function implementations. `default`, `delete`. Monostate. Noncopyable. Structural patterns. Adapter. Bridge. Composite. Decorator. Behavioral patterns. Interface-based programming. Memento. Observer. State. State machines. Cyclic dependencies. Class forward declarations. Undo-redo functions. Strategy. Template method. Non-virtual interface. NVI idiom. Fragile base class. Other patterns.

Generic programming. Standard template library. Dynamic polymorphism. Virtual functions. Static polymorphism. Function templates. Advantages and disadvantages. Curiously recurring template pattern. CRTP. Upside-down inheritance. Banishing virtuality. Extending classes functionality. Counting class instances. Private inheritance and friends. Barton–Nackman trick with operators. Boost.Operators. Mixins. Variadic base classes. Template template parameters.

Educational Materials

- `05.01.pattern.builder`
- `05.02.pattern.factory`
- `05.03.pattern.prototype`
- `05.04.pattern.singleton.monostate.noncopyable`
- `05.05.pattern.adapter`
- `05.06.pattern.bridge`
- `05.07.pattern.composite`
- `05.08.pattern.decorator`
- `05.09.pattern.memento`
- `05.10.pattern.observer`
- `05.11.pattern.state`
- `05.12.pattern.strategy`
- `05.13.pattern.template.method`
- `05.14.template.polymorphism`
- `05.15.pattern.crtp`
- `05.16.pattern.crtp.observer`
- `05.17.pattern.crtp.operator.overload`
- `05.18.pattern.crtp.mixin`
- `05.19.pattern.crtp.mixin.constructor`
- `05.20.pattern.crtp.mixin.inheritance.variadic`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, section 27
- D. Vandevoorde, C++ Templates, 2nd Edition, sections 18, 21, 5.8
- S. Meyers, Effective C++, 3rd Edition, sections 6, 35
- S. Meyers, More Effective C++, sections 25, 26, 32
- S. Meyers, Effective Modern C++, sections 3.5, 4.5
- S. Dewhurst, C++ Gotchas, sections 72, 76, 90, 92
- H. Sutter, C++ Coding Standards, sections 39, 50, 53, 64
- R. Grimm, C++20 Get the Details, section 4.3
- M. Bancila, The Modern C++ Challenge, section 8
- E. Gamma, Design Patterns, sections 1, 3-6
- B. Schaeling, The Boost C++ Libraries, 2nd Edition, section 72

06 Project Organization

Program: 2 or 3 lectures: duration up to 8 hours in total; 2 programming problems for homework.

Abstract: Source and header files. Build stages. Namespaces. Modules. Boost. Static and dynamic libraries.

Multi-file programs. Source and header files. Declarations and definitions. ODR. Identifiers visibility. Build stages. 9 phases of translation. Preprocessing. Preprocessors. Code generation. Preprocessor directives. `include`. Search directories. Transitive includes. Macro definitions. Object-like and function-like macros. `define`, `undef`. Multi-line macros. Macro problems. Side effects. Predefined macros. `DEBUG`, `NDEBUG`. Standard predefined macros and identifiers. `FILE`, `LINE`, `DATE`, `TIME`. `func`. `std::source_location`. Conditional compilation. `if`, `defined`, `else`, `endif`. Header guards. `pragma once`. Disabling warnings. `pragma warning`. Trigraphs. Translation units. Compilation. Compilers. Changes and partial recompilation. Reducing compile-time dependencies. Pointer to implementation. PImpl idiom. Runtime and maintenance overheads. Precompiled header files. Object files. Library files. Linkage. Linkers. External and internal linkage. Linker errors. Multiple defined symbols. Unresolved external symbols. Global variables and constants. `extern`. Anonymous namespaces. Class and template definitions in headers. Exceptions from ODR. Namespaces. `namespace`. Collisions of identifiers. Namespace scopes. Global namespace. Scope resolution operator. Qualified names. Argument-dependent lookup. Koenig's search. Namespaces additivity. Nested namespaces. Namespace aliases. Inline namespaces. System versions. Inline variables. `inline`. Namespace std. Using declarations. `using`.

Build automation systems. CMake. CMakeLists. Projects. Standard requirements. Variables. Lists. Foreach. Packages. Include and link directories. Targets. Executables. Libraries. Properties. Compile and link flags. Compile options and definitions. Preparing and building sources. Shell scripts. Bash. Package managers. Modules. Module declarations. `module`. Module fragments. Global module fragments. Private module fragments. Module interface and implementation units. Exporting declarations and definitions. Single, group and namespace exports. `export`. Importing modules. `import`. Classes and templates in modules. Submodules and partitions. Modules linkage. Standard library modules.

Boost. Building and using Boost. Project properties. Additional include and library directories. Visual Studio projects. Static libraries. Interfaces in header files. Implementations in library files. Dynamic link libraries. Implicit and explicit libraries linkage. Exporting and importing symbols and aliases. `extern "C"`. Boost.DLL. Runtime libraries swapping.

Educational Materials

- `06.01.project.translation`
- `06.02.project.header`
- `06.03.project.source`
- `06.04.project.source.main`
- `06.05.pattern.pimpl`
- `06.06.project.header.precompiled`
- `06.07.project.source.precompiled`
- `06.08.project.tool.cmake`
- `06.09.project.module`
- `06.10.project.module.source`
- `06.11.project.module.submodule`
- `06.12.project.library`
- `06.13.project.library.static`
- `06.14.project.library.shared`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 12.6, 14, 15
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 3.2-3.4
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 4.1, 4.2
- D. Vandevoorde, C++ Templates, 2nd Edition, sections 9.3, 13.2, A
- S. Meyers, Effective C++, 3rd Edition, sections 2, 4, 31
- S. Dewhurst, C++ Gotchas, sections 3, 8, 25-28, 55
- H. Sutter, C++ Coding Standards, sections 10, 16, 21-24, 43, 57-59, 61, 63
- R. Grimm, C++20 Get the Details, section 4.2

07 Errors and Exceptions

Program: 2 or 3 lectures: duration up to 8 hours in total; 6 programming problems for homework.

Abstract: Errors. Assertions. Terminations. Utilities. Exceptions. Debugging. Logging. Testing. Profiling.

Errors handling. Invariants. Pre- and postconditions. Assertions. `assert`. Compile-time assertions. `static_assert`. Runtime errors. Repeats. Normal terminations. `std::exit`, `std::atexit`. Abnormal terminations. `std::abort`. Return codes. Global variables. Additional arguments. Return code inconveniences. Unions. `union`. Union members lifetime. Anonymous unions. Variadic values. `std::variant`. Optional values. `std::optional`. Tri-state boolean.

Exceptions. Standard exceptions hierarchy. `std::exception`. `std::system_error`. `std::error_code`. User-defined exceptions. Throw statements. `throw`. Rethrowing exceptions. `[[noreturn]]`. Try sections. `try`. Function try blocks. Catch handlers. `catch`. Catch-all handlers. `std::exception_ptr`. `std::current_exception`. Catch handlers order. `std::cerr`. Exceptions in constructors and destructors. Nothrow specifier. `noexcept`. `std::declval`. Outdated dynamic exception specifications. Forwarding exceptions. Exception problems and downsides. Exception conversions. Stack unwinding. Zero-overhead principle. Unhandled exceptions. `std::terminate`. Exception safety guarantees. No guarantees. Basic, strong, no-throw and no-fail guarantees. Design of exception-safe stack interface.

Debugging. Detecting and reproducing problems. Debugging strategies. Commenting. Printing. Logging. Boost.Log. Preprocessing. Unit testing. Writing unit tests. Boost.Test. Usage variants. Test trees. Master tests. Test suites. Test cases. Data driven test cases. Datasets and samples. Operations on datasets. Template test cases. Boost.MPL. Parametrized test cases. Test fixtures. Program states. Integrated debugger. Debug mode. Stepping. Step into, over and out. Run to cursor. Start and continue. Breakpoints. Watching variables. Call stack. Profiling. Integrated profiler. Release mode. Processor usage. Memory usage. Memory leaks. Heap snapshots. Code refactoring and optimization. Google instruments. Time measurement. Google.Benchmark. Output formats. Statistical stability. Passing arguments. Calculating complexity. Preventing optimization. Exiting with an error. Helpful macros. Writing unit tests. Google.Test. Test cases. Assertions. Expectations. Test fixtures. Parametrized test cases. Invoking tests. Mocking framework classes.

Educational Materials

- `07.01.error.assertion.termination`
- `07.02.class.union`
- `07.03.utility.variant.optional`
- `07.04.error.exception`
- `07.05.error.exception.stack.interface`
- `07.06.project.tool.boost.logging`
- `07.07.project.tool.boost.testing`
- `07.08.project.tool.profiler.cpu`
- `07.09.project.tool.profiler.memory`
- `07.10.project.tool.google.benchmark`
- `07.11.project.tool.google.testing`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 8.3, 13.1, 13.2, 13.4, 13.5, 30.4
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 2.4, 3.5, 13.5
- N. Josuttis, The C++ Standard Library, 2nd Edition, section 4.3
- D. Vandevoorde, C++ Templates, 2nd Edition, section 26
- S. Meyers, Effective C++, 3rd Edition, sections 8, 29
- S. Meyers, More Effective C++, sections 9-15
- S. Meyers, Effective Modern C++, section 3.8
- S. Dewhurst, C++ Gotchas, sections 64, 65
- H. Sutter, C++ Coding Standards, sections 14, 51, 62, 68-75, 97
- B. Schaeling, The Boost C++ Libraries, 2nd Edition, section 62

08 Numbers Processing

Program: 2 or 3 lectures: duration up to 6 hours in total; 6 programming problems for homework.

Abstract: Bits and bytes. Manipulators. Enumerations. Floating-point numbers. Random numbers. Time.

Manipulators. `std::showbase`. `std::oct`, `std::dec`, `std::hex`, `std::boolalpha`. Binary, octal, decimal and hexadecimal literals. Bitwise operators. Bit shifts arithmetic. Fixed width integer types. `std::int#_t`, `std::uint#_t`. Manipulating bits and bytes. Storage alternatives. `std::bitset`. `std::byte`. `std::to_integer`. `std::as_bytes`. Endianess. Little and big endian. `std::endian`. Type punning. Reinterpreting bytes. `std::bit_cast`. Requirements and constraints. `reinterpret_cast`. Safety aspects. Additional bit functions. `std::bit_ceil`, `std::bit_floor`. Bit fields. Size. Alignment. Binary Gray code. Encoding and decoding. Indian algorithm. Overflow. Enumerations. `enum`. Scoped enumerations. Scopes and implicit conversions. Underlying types. Unscoped enumerations. Namespaces pollution. Floating-point numbers. Scientific notation. Precisions. `std::scientific`, `std::defaultfloat`, `std::fixed`. `std::setprecision`. Output formats. `std::setw`, `std::right`, `std::left`, `std::setfill`. Minimum and maximum values. Infinity, NaN and negative zero. `std::numeric_limits`, `errno`. Floating-point numbers comparison. Epsilon constants. Implementations and optimizations by Donald Knuth. Extending floating-point numbers precision. Boost.Multiprecision. Additional math functions. Boost.Math. Complex numbers. `std::complex`.

Pseudo-random numbers generation. Random seeds. Entropy sources. Non-deterministic random number generators. `std::random_device`. Random number engines. Period, probability and performance. 32-bit and 64-bit Mersenne twisters by Matsumoto and Nishimura. `std::mt19937`. Random number distributions. Uniform, Bernoulli, Poisson, normal and sampling distributions. `std::normal_distribution`. Random library from C. `std::srand`, `std::rand`. Monte-Carlo methods. Calculating Pi constant. `std::uniform_real_distribution`. Time management. Chrono library. `std::chrono`. Clocks. Epochs. Unix epoch. Ticks. System clock. Steady clock. Other clocks. Time points. Durations. Code timings. Timer class. Calendar dates and time. Conversions to time points and durations. Date arithmetic. Weekdays. Time zones. Standard literals. `std::literals`. User-defined literals. Numeric literal operators.

Educational Materials

- `08.01.number.integer`
- `08.02.algorithm.binary.gray.code`
- `08.03.algorithm.indian.exponentiation`
- `08.04.class.enumeration`
- `08.05.number.floating.point`
- `08.06.number.floating.point.multiprecision`
- `08.07.number.complex`
- `08.08.number.random.generation`
- `08.09.algorithm.monte.carlo`
- `08.10.algorithm.monte.carlo.simplex`
- `08.11.time.chrono`
- `08.12.time.chrono.calendar`
- `08.13.implementation.operator.literal`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 6.2, 8.4, 35.2, 40.2-40.5, 40.7
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 2.5, 13.4, 13.7, 14
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 5.3, 5.5, 5.7, 12.5, 17, S.1, S.2
- S. Meyers, More Effective C++, section 2
- S. Meyers, Effective Modern C++, section 3.4
- H. Sutter, C++ Coding Standards, section 92
- R. Grimm, C++20 Get the Details, sections 5.4, 5.5
- M. Bancila, The Modern C++ Challenge, sections 1.10, 1.13, 2.17, 2.22, 5

09 Memory Management

Program: 3 or 4 lectures: duration up to 12 hours in total; 16 programming problems for homework.

Abstract: Resources. RAII. Smart pointers. Iterators. Memory. Addresses. Alignment. Allocations. Allocators.

Memory management. Garbage collectors. Resource acquisition is initialization. RAII idiom. Smart pointers. Shared ownership. `std::shared_ptr`. Counting references. Internal counters. `std::make_shared`. Hidden advantages. Dealing with built-in arrays. Destruction policies. `std::default_delete`. Cyclic dependencies. `std::weak_ptr`. Strong and weak references. Exclusive ownership. `std::unique_ptr`. `std::make_unique`. Sources and sinks of data. `std::enable_shared_from_this`. Iterators. Categories of iterators. Input, output, forward, bidirectional and random access iterators. Iterator traits and tags. Generic functions for iterators. `std::advance`, `std::next`, `std::prev`, `std::distance`, `std::iter_swap`. Iterator adaptors. Reverse iterators. Facade. Facades for iterators. Boost.Iterator.

Processes. Memory. Virtual address space. Physical address space. Addresses translation. Page tables. Page table entries. Memory management units. Translation lookaside buffers. Translation failures. System kernel. Kernel segment. Page fault. Free space. Stack. Stack overflow. Memory mapping segment. Heap. Memory manager. Data segment. Code segment. Segmentation fault. Objects storage durations. Lifetime of automatic, dynamic, static, temporary and thread-local objects. Miscellaneous. Memory access granularity. Data alignment. `alignof`, `alignas`. Failed allocations handling. `std::set_new_handler`. Nothrow new. `std::nothrow`. Uninitialized memory allocations. `operator new`, `operator delete`. Placement new. Explicitly called destructors. Overloading memory management functions. Allocators. Standard allocators. `std::allocator`. Allocator traits. `std::allocator_traits`. Costs of allocations. Memory fragmentation. Custom allocators. Linear allocators. `std::align`, `std::max_align_t`, `std::align_val_t`. Arena, stack, chain, block and no heap allocators. Benchmarks. Memory resources. Polymorphic allocators. `std::pmr`.

Educational Materials

- 09.01.pointer.smart
- 09.02.pointer.smart.shared.this
- 09.03.implementation.pointer.smart.unique
- 09.04.implementation.pointer.smart.shared
- 09.05.iterator.classification
- 09.06.implementation.iterator.forward.list
- 09.07.implementation.iterator.forward.list.facade
- 09.08.pattern.facade
- 09.09.memory.address.space
- 09.10.memory.address.translation
- 09.11.memory.alignment
- 09.12.memory.operator.new.handler
- 09.13.memory.operator.new.placement
- 09.14.implementation.utility.optional
- 09.15.memory.allocator.standard
- 09.16.memory.allocator
- 09.17.memory.allocator.block
- 09.18.implementation.allocator.arena
- 09.19.implementation.allocator.stack
- 09.20.implementation.allocator.chain
- 09.21.implementation.allocator.block
- 09.22.implementation.allocator.fixed
- 09.23.implementation.allocator.arena.adaptor
- 09.24.memory.allocator.polymorphism

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 33.2-33.4, 34.3-34.6
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 12.2, 12.3, 13.2, 13.6
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 4.6, 5.2, 9, 19, S.3
- S. Meyers, Effective C++, 3rd Edition, sections 13-15, 17, 47, 49-52
- S. Meyers, More Effective C++, sections 4, 8, 27-29
- S. Meyers, Effective Modern C++, sections 4.1-4.4
- S. Dewhurst, C++ Gotchas, sections 24, 29, 43, 61-63, 67, 68
- H. Sutter, C++ Coding Standards, sections 13, 45, 46, 60
- M. Bancila, The Modern C++ Challenge, section 2.21

10 Containers of Objects

Program: 2 or 3 lectures: duration up to 8 hours in total; 5 programming problems for homework.

Abstract: Sequential containers. Multidimensional arrays. Container adaptors. Sets and maps. Hash tables.

Containers. Semantics of values, pointers and references. Exceptions safety. Order of elements. Sequential containers. Dynamic containers. `std::vector`. Size and capacity. Accessing elements. Free functions. `std::size`, `std::ssize`. Non-constant iterators. `std::begin`, `std::end`. Constant iterators. `std::cbegin`, `std::cend`. Dummy element after the last one. Copying and moving collections. Move iterators. `std::make_move_iterator`. Inserting and erasing elements. Constructors. In-place construction. Braced initializer lists. `std::initializer_list`. Types inference problem. Operations complexity. Deque containers. `std::deque`. Static containers. `std::array`. `std::to_array`. Doubly-linked list containers. `std::list`. Singly-linked list containers. `std::forward_list`. Obtaining the middle node of lists. Iterators invalidation. Vector specialization for booleans. Proxy. Remote, virtual, security and caching proxies. Smart pointers. Lazy loading. Multidimensional arrays. Row-major and column-major orders. Memory layout. Flattening arrays. Boost.MultiArray. Vectors, matrices and tensors. Operating vectors and matrices. Boost.uBLAS. Numeric arrays. `std::valarray`. Container adaptors. Last in - first out. LIFO. `std::stack`. First in - first out. FIFO. `std::queue`. FIFO with priority. `std::priority_queue`. Circular buffer. Boost.CircularBuffer. Linearization.

Balanced binary search trees. Red-black trees. Black height. Sets. `std::set`, `std::multiset`. Strict weak ordering. Antisymmetry, transitivity, irreflexivity and transitivity of equivalence. Operations complexity. Inserting, erasing and finding elements. Hints. Binary search. Extracting nodes. Associative arrays. Maps. `std::map`, `std::multimap`. Replacing keys. `std::pair`, `std::make_pair`. Unordered containers. Hash functions. `std::hash`. Combined hash. Boost.ContainerHash. Hash tables. `std::unordered_#`. Objects and buckets. Hashing by division. Load factor. Collisions resolution. Separate chaining. Iterators of hash tables. Open addressing. Rehashing. Operations complexity. Hashers. Simultaneous interfaces. Boost.MultiIndex. Biassociative arrays. Boost.Bimap. Flyweight. Boost.Flyweight.

Educational Materials

- `10.01.container.vector.deque.array.list.memory`
- `10.02.container.vector.deque.array.list`
- `10.03.container.vector.deque.array.list.sorting`
- `10.04.pattern.proxy.vector`
- `10.05.container.multidimensional`
- `10.06.container.multidimensional.multiarray`
- `10.07.container.multidimensional.ublas`
- `10.08.container.multidimensional.valarray`
- `10.09.algorithm.longest.common.subsequence`
- `10.10.container.adaptor`
- `10.11.container.adaptor.stack`
- `10.12.container.circular.buffer.memory`
- `10.13.container.circular.buffer`
- `10.14.container.red.black.tree.memory`
- `10.15.container.red.black.tree`
- `10.16.implementation.function.hash`
- `10.17.container.hash.table.memory`
- `10.18.container.hash.table`
- `10.19.implementation.class.functor.hash.equal`
- `10.20.container.multiindex.bimap`
- `10.21.pattern.flyweight`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 11.3, 13.4, 29, 31, 34.2
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 11, 13.4
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 5.1, 7, 8, 12
- S. Meyers, More Effective C++, section 30
- S. Meyers, Effective Modern C++, sections 2.2, 3.1, 3.7, 8.2
- S. Dewhurst, C++ Gotchas, section 34
- H. Sutter, C++ Coding Standards, sections 76, 78-83
- R. Grimm, C++20 Get the Details, section 5.3
- B. Schaeling, The Boost C++ Libraries, 2nd Edition, sections 12, 13, 16, 19, 66

11 Algorithms on Ranges

Program: 2 or 3 lectures: duration up to 8 hours in total; 13 programming problems for homework.

Abstract: Function objects. Lambda expressions. Visitors. Standard algorithms. Ranges and views. Graphs.

Types of functions. Function pointers. Passing functions as arguments. Templates with perfect forwarding. Functions invocation. `std::invoke`. Callable objects. Internal states. Functions. Static variables. Functors. Function objects. Data members. Overloading parenthesis operator. Operator function objects. Predicates. `std::less`, `std::greater`. `std::negate`, `std::plus`. Lambda expressions. Closures. Capture lists. Move captures. Generalized lambda captures. `mutable`. Type erasure. `std::function`. Bindings. `std::bind`. Templated lambdas. Lambdas in STL. Class instance pointer capture. Command. Visitor. Double dispatch. Advanced visitors. `std::visit`. Heterogeneous vector. Variadic integer indexes sequences. `std::integer_sequence`, `std::make_integer_sequence`. Cartesian product.

Iterator adaptors. Insert iterators. Front and back inserters. `std::front_inserter`, `std::back_inserter`. General inserters. `std::inserter`. Standard algorithms. Batch operations. Search operations. Copy, move and swap operations. Transform operations. Generate operations. Remove and erase operations. Order-changing operations. Sample operations. Partition operations. Sort operations. Binary search operations. Merge operations. Set operations. Heap operations. Maximum and minimum operations. Lexicographical comparison operations. Permutation operations. Numeric operations. Uninitialized memory operations. Range-based for loop. Bindings. Algorithms on ranges. Ranges. `std::ranges`. Views. `std::views`. Projections. Adaptors and algorithms. Boost graph library. Boost.Graph. Adjacency lists. Incidence matrices. Vertices. Edges. Directions. Properties. Visitors. Breadth first search. BFS. BFS visitors. Depth first search. DFS. DFS visitors. Edge weights. Dijkstra shortest paths. AT&T Graphviz utilities.

Educational Materials

- `11.01.pointer.function`
- `11.02.class.functor`
- `11.03.class.functor.lambda.expression`
- `11.04.pattern.command`
- `11.05.pattern.visitor`
- `11.06.pattern.visitor.utility.variant`
- `11.07.pattern.visitor.utility.any`
- `11.08.implementation.utility.variant`
- `11.09.template.metaprogramming.cartesian.product`
- `11.10.algorithm.iterator.adaptor.inserter`
- `11.11.algorithm.standard`
- `11.12.algorithm.standard.range.view`
- `11.13.algorithm.graph`
- `11.14.algorithm.graph.example`
- `11.15.algorithm.graph.traversal.bfs`
- `11.16.algorithm.graph.traversal.dfs`
- `11.17.algorithm.graph.search.dijkstra`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 11.4, 12.5, 20.6, 32, 33.5, 33.6, 40.6
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 6.3, 12.5-12.8, 13.8
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 4.4, 9.4, 10, 11
- D. Vandevoorde, C++ Templates, 2nd Edition, sections 11.1, 22
- S. Meyers, Effective Modern C++, section 6
- H. Sutter, C++ Coding Standards, sections 84-89
- R. Grimm, C++20 Get the Details, sections 4.7, 5.1, 5.3
- M. Bancila, The Modern C++ Challenge, section 2.20
- B. Schaeling, The Boost C++ Libraries, 2nd Edition, section 31
- J. Siek, The Boost Graph Library, sections 1-16

12 Text Data Processing

Program: 2 or 3 lectures: duration up to 8 hours in total; 14 programming problems for homework.

Abstract: Characters. Encodings. Strings. Views. Locales. Facets. Regular expressions. Grammars. Parsers.

Characters. `char`. ASCII codes. Signed and unsigned char types. Escape sequences. Encodings. Multibyte and wide encodings. 7-bits and 8-bits ASCII. ISO-Latin-1. Unicode. Unicode literals. Universal character set. Universal transformation format. UTF-8, UTF-16, UTF-32. `char8_t`, `char16_t`, `char32_t`, `wchar_t`. Endianness. Big and little endian. `std::endian`. Byte order mark. BOM. Encodings problems. Strings. `std::basic_string`, `std::char_traits`. Aliases for strings. `std::string`, `std::u8string`, `std::u16string`, `std::u32string`, `std::wstring`. Reading strings. `std::getline`. Ignoring space characters. `std::ws`. `std::quoted`. Standard string literals. Indexes and iterators. Member and free functions. `std::stoi`, `std::to_string`. C-strings. Char arrays. Null-terminated strings. C-strings library. `std::strlen`. Case-insensitive strings. Small strings optimization. SSO. Views. `std::string_view`. Advantages and hazards. Internationalization. I18N. Localization. L10N. Locales. `std::locale`, `setlocale`. Classic C-locale. Using locales. Facets. `std::use_facet`. Punctuation and currency. `std::numpunct`, `std::moneypunct`. Dates and times. `std::time_put`, `std::time_get`. Converting encodings to, from and between UTFs. Boost.Locale.

Regular expressions. `std::regex`. ECMA-script grammar. Character groups, classes and quantities. Patterns. Raw strings. Matching patterns. `std::regex_match`. Searching patterns. `std::regex_search`. `std::match_results`. Prefixes and suffixes. Regular expression iterators. `std::regex_iterator`. Tokens. `std::regex_token_iterator`. Replacing patterns. `std::regex_replace`. Regular expression flags. `std::regex_constants`. Grammars. Recursive descent parsers. Calculator by B. Stroustrup. Statements, declarations, expressions, terms and primaries. Streams and tokens. Extended Backus Naur form. EBNF. Parsing expression grammars. Boost.Spirit. Phrase parsers. Rules and sequences. Attributes. Parser for complex numbers. Parser for roman numbers. Symbol parsers. Using namespaces and macros. Parser for structures. Adaptors. Boost.Fusion. Advanced calculator. Abstract syntax trees. AST. Visitors.

Educational Materials

- `12.01.character.encoding`
- `12.02.character.encoding.unicode.utf`
- `12.03.character.string.view`
- `12.05.locale.facet`
- `12.06.locale.facet.utf.converter`
- `12.07.parser.regex.ecma.script`
- `12.08.parser.regex`
- `12.09.parser.grammar.arithmetic`
- `12.10.parser.grammar.arithmetic.calculator`
- `12.11.parser.spirit.number.integer`
- `12.12.parser.spirit.number.complex`
- `12.13.parser.spirit.number.romanus`
- `12.14.parser.spirit.fusion.class.structure`
- `12.15.parser.spirit.grammar.arithmetic.calculator`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 36, 37, 39
- B. Stroustrup, A Tour of C++, 2nd Edition, section 9
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 13, 14, 16
- R. Grimm, C++20 Get the Details, section 5.6
- M. Bancila, The Modern C++ Challenge, section 3

13 Streams and Formats

Program: 2 or 3 lectures: duration up to 6 hours in total; 5 programming problems for homework.

Abstract:

IOStream library. Stream classes hierarchy. `std::ios_base`. Stream states. `goodbit`, `eofbit`, `failbit`, `badbit`. Input validation. Exceptions. Formatting flags. Manipulators. `std::endl`. User-defined manipulators. Overloading input and output operators. `std::basic_ios`. Template specializations. Stream buffers. `std::basic_streambuf`. Input and output streams. `std::basic_istream`, `std::basic_ostream`, `std::basic_iostream`. Virtual inheritance. Global stream objects. `std::cin`, `std::cout`, `std::cerr`, `std::clog`. Standard header files. Stream classes for files. `std::fstream`. File opening modes. Random file access. Stream classes for strings. `std::stringstream`. Character-by-character input data processing. Input and output stream buffer iterators. `std::istreambuf_iterator`, `std::ostreambuf_iterator`. Breaking input. End of file character. EOF. Streams library from C. `stdin`, `stdout`, `stderr`. `std::clearerr`. Synchronization with stdio. Input and output stream iterators. `std::istream_iterator`, `std::ostream_iterator`. Formatting library. `std::format`, `std::format_to`. Python notation. `std::formatter`.

Educational Materials

- `13.01.stream.class.inheritance`
- `13.02.stream.class`
- `13.03.stream.class.file`
- `13.04.stream.class.string`
- `13.05.stream.iterator.buffer.input.output`
- `13.06.algorithm.erase.comment`
- `13.07.algorithm.erase.comment.test`
- `13.08.stream.output.format`

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, section 38
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 10, 12.4
- N. Josuttis, The C++ Standard Library, 2nd Edition, section 15
- S. Meyers, More Effective C++, section 25
- M. Bancila, The Modern C++ Challenge, sections 3.23, 4, 9
- B. Kernigan, The C Programming Language, 3nd Edition, task 1.23

14 Parallel Programming

Program: 3 or 4 lectures: duration up to 12 hours in total; 0 programming problems for homework.

Abstract:

Educational Materials

-

Supporting Resources

- B. Stroustrup, The C++ Programming Language, 4th Edition, sections 41, 42
- B. Stroustrup, A Tour of C++, 2nd Edition, sections 12.9, 15
- N. Josuttis, The C++ Standard Library, 2nd Edition, sections 4.5, 18
- S. Meyers, Effective Modern C++, sections 3.10, 7
- H. Sutter, C++ Coding Standards, section 12
- R. Grimm, C++20 Get the Details, sections 6, 7

15 Network Technologies

Program: 3 or 4 lectures: duration up to 12 hours in total; 0 programming problems for homework.

Abstract:

Educational Materials

-

Supporting Resources

-

16 Support Embeddings

Program: 2 or 3 lectures: duration up to 6 hours in total; 0 programming problems for homework.

Abstract: Assemblers. Registers. Instructions. Floating point operations. Long arithmetic. Python embedding.

Assemblers. Intel x86 assembly language. Microsoft Macro Assembler dialect. Memory, registers and disassembled code. Stack. Calling conventions. `cdecl`, `pascal`, `stdcall`. Callers and callees. Arguments passing order. Return values. Stack purging. Inline assembler code. `asm`. General-purpose registers. `eax`, `ebx`, `ecx`, `edx`. Index registers. `esi`, `edi`. Stack pointers. `esp`. Frame pointers. `ebp`. Data movement instructions. `push`, `pop`, `mov`. Arithmetic instructions. `add`, `sub`, `inc`, `dec`. Control flow instructions. `jmp`, `cmp`, `call`. Conditional jumps. Labels. Addressing memory. Floating point assembly language. Floating point processors. Floating point stack, registers and instructions.

Long arithmetic. Digits representations. Overloading arithmetic operators. Fast Karatsuba multiplication. Other operations. Boost.Multiprecision. Python embedding. Dealing with interpreter. Python C/C++ API. Boost.Python. Global interpreter locker. GIL. Threads and processes. Importing functions. Calling functions. Passing arguments. Extracting results. Handling exceptions. Python tools. Math package. Calculating factorial. Matplotlib package. Visualizing data.

Educational Materials

- `16.01.embedding.assembler`
- `16.02.implementation.class.big.integer`
- `16.03.number.integer.multiprecision`
- `16.04.embedding.python.wrapper.header`
- `16.05.embedding.python.wrapper.source`
- `16.06.embedding.python.caller`
- `16.07.embedding.python.script`

Supporting Resources

- R. Hyde, The Art of Assembly Language, 2nd Edition, sections 1-6