Software Engineering

Moscow Institute of Physics and Technology

Table of Contents

01. Introduction and Brief Overview

General Introduction

- `01.01` – Software engineering. The C++ programming language. Programming paradigms. Instruments. References.

Environment and Compiler

- `01.02` – Environment Visual Studio Code. Terminal. Compiler g++ from GCC. Minimal program. Function `main`.

Standard Library

- `01.03` – Standard library overview. Comments. Documenting code. Utility Doxygen.
- `01.04` – Standard library header files.

Version Control System

- `01.05` – Version control system Git. Project hosting system GitHub. Git graphical client SmartGit.

02. Basics of Programming

Fundamental Data Types

- 02.01 – Type `bool`. Automatic objects. Literals `false` and `true`. Operator `sizeof`. Attribute `maybe_unused`.
- 02.02 – Type `char`. Escape sequences. Portability problem.
- 02.03 – Type `int`. Modifiers `short` and `long`. Literal suffixes. Two's complement. Overflow problem.
- 02.04 – Types `float`, `double` and `long double`. Floating point formats. Precision problem.
- 02.05 – Type signness. Modifiers `signed` and `unsigned`.
- 02.06 – Constants. Qualifier `const`.
- 02.07 – Type aliases. Declaration `using`. Specifier `typedef`. Alias `std::size_t`. Fixed-width integer types.

Objects and Variables

- 02.08 – Declarations. Definitions. Default, value, direct, copy and list initialization. Undefined behavior.
- 02.09 – Type inference. Placeholder `auto`.
- 02.10 – Implicit, explicit, narrowing and C-style type conversions. Operator `static_cast`. Temporary objects.

Operators and Expressions

- 02.11 – Logical operators. Alternative representations. Short-circuit evaluations.
- 02.12 – Expressions. Arithmetic and comparison operators. Operator arity. Operator precedence.
- 02.13 – Operator exclusive or.
- 02.14 – Assignment operators. Arithmetic swap algorithm. Function `std::swap`.
- 02.15 – Operator division. Operator remainder.
- 02.16 – Evaluation order. Unspecified behavior. Operator associativity. Operator comma.

Selection Statements

- 02.17 – Statement `if`. Utility Compiler Explorer.
- 02.18 – Statement `switch`. Labels `case` and `default`. Attributes `fallthrough`, `likely` and `unlikely`.
- 02.19 – Ternary operator.

Loops and Jump Statements

- 02.20 – Statement `for`.
- 02.21 – Statement `continue`.
- 02.22 – Statement `break`. Infinite loops.
- 02.23 – Statement `goto`. Labels.
- 02.24 – Statement `while`.
- 02.25 – Statement `while-do`.

Memory Management

- 02.26 – Pointers. Operator address of. Operator dereference. Literal `nullptr`.
- 02.27 – Constant pointers. Pointers to constants.

Collections and Containers

- 02.28 – Static arrays. Aggregate initialization. Non-standard compiler extensions. Stack limit. Command ulimit.
- 02.29 – Function `std::size`. Index access operator. Pointer arithmetic.
- 02.30 – Dynamic objects. Operators `new` and `delete`. Dynamic arrays. Operators `new[]` and `delete[]`.
- 02.31 – Container `std::vector` overview. Complexities `O(1)` and `O(N)`. Amortized complexity.

Lvalue References

- 02.32 – Lvalue references.
- 02.33 – Constant lvalue references.
- 02.34 – Lvalue reference type inference. Placeholder `decltype(auto)`. Specifier `decltype`.
- 02.35 – Wrapper `std::reference_wrapper`.

Functional Programming

- 02.36 – Functions. Forward declarations. Statement `return`. Attribute `nodiscard`. Object `std::ignore`.
- 02.37 – Calling conventions x86. Attributes `cdecl`, `stdcall` and `fastcall`.
- 02.38 – Type `void`. Default arguments.
- 02.39 – Function overloading.
- 02.40 – Passing arguments by value, by lvalue reference and by pointer. View `std::span`.
- 02.41 – Dangling pointers and references. Static objects. Specifier `static`.
- 02.42 – Inline functions. Specifier `inline`. Attribute `noinline`. Special memory. Qualifier `volatile`.
- 02.43 – Recursion. Factorial. Binomial coefficients. Catalan numbers. Trailing return types.
- 02.44 – Insertion and merge sort algorithms. Complexities `O(N*N)` and `O(N*log(N))`. Function `std::midpoint`.
- 02.45 – Binary search algorithm. Complexity `O(log(N))`.

03. Object - Oriented Programming

User - defined Data Types

- 03.01 – Structures. Declaration `struct`. Data members. Instances. Designated initialization.
- 03.02 – Operator point. Operator arrow.
- 03.03 – Classes. Declaration `class`. Encapsulation. Specifiers `public` and `private`. Constructors. Destructors.
- 03.04 – Member functions. Constant member functions.
- 03.05 – Nested classes. Pointer `this`. Logical and bitwise constancy. Specifier `mutable`.
- 03.06 – Static members.

Interclass Relations

- 03.07 – Composition, aggregation, association and dependency relations.
- 03.08 – Friend functions and classes. Specifier `friend`.
- 03.09 – Pattern Attorney - Client.
- 03.10 – Pattern PassKey.

Inheritance and Hierarchies

- 03.11 – Class hierarchies. Base and derived classes. Public inheritance. Protected members. Specifier `protected`.
- 03.12 – Private inheritance. Composition.
- 03.13 – Multiple inheritance. Virtual inheritance. Diamond problem.
- 03.14 – Appendix: scheme.
- 03.15 – Empty classes. Empty base optimization. Attribute `no_unique_address`.

Dynamic Polymorphism

- 03.16 – Virtual functions. Specifiers `virtual`, `override` and `final`. Virtual destructors.
- 03.17 – Pure virtual functions. Abstract base classes.
- 03.18 – Virtual pointers. Virtual tables.
- 03.19 – Appendix: scheme.
- 03.20 – Covariant return types.

Runtime Type Identification

- 03.21 – Downcasting type conversions. Operator `dynamic_cast`.
- 03.22 – Operator `typeid`.
- 03.23 – Library Boost.TypeIndex.
- 03.24 – Type `std::any`. Function `std::make_any`. Pointers to raw memory.

Rvalue References

- 03.25 – Lvalue, glvalue, xvalue, rvalue and prvalue expressions.
- 03.26 – Rvalue references.
- 03.27 – Extending temporary objects lifetime.
- 03.28 – Copy and move semantics. Function `std::move`.
- 03.29 – Member function reference qualifiers.
- 03.30 – Container `Vector`. Special member functions. Deep and shallow copy. Copy and swap. Rules of 0, 3 and 5.
- 03.31 – Copy elision. Return value optimization. Named return value optimization.

Operator Overloading

- 03.32 – Rational arithmetic. Type `Rational`. User - defined type conversions. Specifier `explicit`.
- 03.33 – Library Boost.Rational.
- 03.34 – Three - way comparison operator. Strong ordering. Equivalence. Equality.
- 03.35 – Weak ordering.
- 03.36 – Unordered objects. Partial ordering.
- 03.37 – Input and output operators.
- 03.38 – Constant and non - constant index access operators. Constancy type conversions. Operator `const_cast`.

04. Generic Programming

Function Templates

- 04.01 – Function templates. Declaration `template`. Type template parameters. Specifier `typename`. Instantiating.
- 04.02 – Full specializations. Function template overloading.
- 04.03 – Dimov - Abrahams example.
- 04.04 – Non - type template parameters. Passing static arrays by lvalue reference.
- 04.05 – Variadic templates. Template and function parameter packs. Ellipsis. Operator `sizeof...`
- 04.06 – Variadic expressions. Fold expressions. Arithmetic reduce algorithm.
- 04.07 – Tree traverse algorithm. Pointers to members.

Class Templates

- 04.08 – Class templates. Container `Stack`. Default types. Instantiating member functions.
- 04.09 – Class template argument deduction. Template template parameters.
- 04.10 – Full and partial specializations.
- 04.11 – Instantiating friend functions and operators.

Forwarding References

- 04.12 – Forwarding references. Perfect forwarding. Function `std::forward`.
- 04.13 – Template type inference. Reference collapsing rules.
- 04.14 – Special member function templates. Substitution failure is not an error. Metafunction `std::enable_if`.

Special Templates

- 04.15 – Type alias templates. Container `Array`.
- 04.16 – Variable and constant templates.

Constant Expressions

- 04.17 – Template metaprogramming. Compile - time factorial.
- 04.18 – Compile - time prime number test algorithm.
- 04.19 – Constant expressions. Immediate functions. Specifiers `constexpr`, `consteval` and `constinit`.
- 04.20 – Statement `if constexpr`.
- 04.21 – Hybrid template metaprogramming. Compile - time rational arithmetic.
- 04.22 – Type `Tuple`. Dependent names. Ambiguity problem.
- 04.23 – Type `std::tuple`. Functions `std::make_tuple`, `std::get` and `std::tie`. Structured bindings.

Trait Templates

- 04.24 – Metafunctions `is_same`. Base classes `std::false_type` and `std::true_type`.
- 04.25 – Metafunctions `is_any_of` and `is_all_of`.
- 04.26 – Integral arithmetic types. Metafunctions `is_integral`. Base class `std::integral_constant`.
- 04.27 – Metafunctions `is_array`.
- 04.28 – Metafunctions `add_lvalue_reference` and `add_rvalue_reference`.
- 04.29 – Metafunctions `remove_reference`.
- 04.30 – Metafunctions `is_base_of`. Variadic functions.
- 04.31 – Unevaluated contexts. Function `declval`.
- 04.32 – Metafunctions `is_polymorphic`.
- 04.33 – Metafunctions `is_convertible`.
- 04.34 – Compile - time conditions. Metafunctions `enable_if`.

Concepts and Constraints

- 04.35 – Concepts. Declaration `concept`. Constraints. Concepts `same_as`.
- 04.36 – Expression `requires`. Simple and type requirements. Concepts `sized_range`.
- 04.36 – Compound requirements. Concepts `totally_ordered`.
- 04.37 – Clause `requires`. Concept `std::integral`. Abbreviated function templates.

Variadic Type Lists

- 04.39 – Compile - time type collections. Container `Deque`.

05. Software Architecture Patterns

Generative Patterns

- 05.01 – Pattern Builder.
- 05.02 – Pattern Factory method.
- 05.03 – Pattern Abstract factory.
- 05.04 – Pattern Prototype. Virtual constructors.
- 05.05 – Pattern Singleton. Default and deleted special member functions. Specifiers `default` and `delete`.
- 05.06 – Pattern Noncopyable. Library Boost.Noncopyable.

Structural Patterns

- 05.07 – Pattern Adapter.
- 05.08 – Pattern Bridge.
- 05.09 – Pattern Composite.
- 05.10 – Pattern Decorator.
- 05.11 – Pattern Facade.

Behavioral Patterns

- 05.12 – Pattern Memento.
- 05.13 – Pattern Observer.
- 05.14 – Pattern State. Finite-state machines.
- 05.15 – Pattern Strategy.
- 05.16 – Pattern Template method. Non-virtual interfaces.

Template Patterns

- 05.17 – Static polymorphism. Eliminating virtuality.
- 05.18 – Curiously recurring template pattern.
- 05.19 – Mixin based pattern Singleton.
- 05.20 – Pattern Controller.
- 05.21 – Extending functionality. Barton-Nackman trick. Restricted template expansion.
- 05.22 – Library Boost.Operators.
- 05.23 – Mixin based pattern Memento. Inverted inheritance.
- 05.24 – Variadic base classes.

06. Projects and Libraries

Preprocessing Stage

- 06.01 – Multi-file projects. Build stages. Source and header files. Translation units. Object and executable files.
- 06.02 – Preprocessor. Directives `include`, `define`, `undef`, `if`, `else`, `endif` and `pragma`. Macros.
- 06.03 – Macros `FILE`, `LINE`, `DATE`, `TIME` and others. Identifier `func`.
- 06.04 – Utility `std::source_location`.

Compilation and Linkage

- 06.05 – Conditional compilation. Include guards. One definition rule. Specifier `extern`. Inline variables.
- 06.06 – Precompiled header files.
- 06.07 – Global variables and constants. Anonymous namespaces.
- 06.08 – Internal and external linkage. Multiply defined and unresolved external symbols.
- 06.09 – Reducing compile-time dependencies. Pointers to implementations.
- 06.10 – Class implementation details.
- 06.11 – Appendix: main.
- 06.12 – Namespaces. Declaration `namespace`. Scope operator. Argument dependent lookup. Namespace aliases.

Module Support

- 06.13 – Modules. Declaration `module`. Global module fragments. Exporting symbols. Declaration `export`.
- 06.14 – Interface and implementation units. Standard library modules.
- 06.15 – Submodules.
- 06.16 – Importing modules. Declaration `import`.

Build Automation System

- 06.17 – Builder CMake. File CMakeLists. Packages. Targets. Libraries. Scripts.

Custom Libraries

- 06.18 – Importing symbols and aliases. Library Boost.DLL.
- 06.19 – Static libraries.
- 06.20 – Library implementation details.
- 06.21 – Dynamic libraries. C-style linkage. Declaration `extern "C"`. Implicit and explicit library linkage.

07. Handling Errors and Debugging

Code Interruptions

- 07.01 – Compile-time and runtime assertions. Declaration `static_assert`. Macro `assert`.
- 07.02 – Normal and abnormal exits. Functions `std::atexit`, `std::exit`, `std::abort` and `std::terminate`.

Return Code Handling

- 07.03 – Return codes. Macro `errno`. Function `std::strerror`.
- 07.04 – Scoped and unscoped enumerations. Declaration `enum`. Underlying types.
- 07.05 – Unions. Declaration `union`.
- 07.06 – Hybrid return codes. Type `Alternative`. Anonymous unions.
- 07.07 – Type `std::variant`. Type `std::monostate`. Functions `std::get`, `std::visit` and others.
- 07.08 – Ternary logic. Library Boost.Tribool.
- 07.09 – Type `std::optional`. Object `std::nullopt`. Function `std::make_optional`.
- 07.10 – Type `std::expected`. Type `std::unexpected`.

Exception Handling

- 07.11 – Statements `throw`, `try` and `catch`. Stack unwinding. User-defined exceptions. Attribute `noreturn`.
- 07.12 – Exception safety guarantees. Specifier and operator `noexcept`. Zero-overhead principle.
- 07.13 – Exception safe container `Stack` interface.
- 07.14 – Backtracing. Call stack. Container `std::stacktrace`.

Debugging and Profiling

- 07.15 – Debugging. Debugger GDB. Commands run, continue, next, step, break, print, list and backtrace.
- 07.16 – Memory profiling. Internal compiler sanitizers.
- 07.17 – External sanitizers Valgrind.
- 07.18 – Performance profiling. Profiler Callgrind. Visualizer KCachegrind.
- 07.19 – Logging. Library Boost.Log.
- 07.20 – Library Google.Log.
- 07.21 – Testing. Unit tests. Test suites and cases. Datasets. Fixtures. Library Boost.Test.
- 07.22 – Assertions. Expectations. Library Google.Test.
- 07.23 – Microbenchmarking. Complexity evaluation. Library Google.Benchmark.

08. Instruments of Calculus

Bitwise Processing

- 08.01 – Number systems. Binary, octal, decimal and hexadecimal literals.
- 08.02 – Bitwise logical operators. Bitwise swap algorithm.
- 08.03 – Reflected binary Gray code. Gray code encode and decode algorithms.
- 08.04 – Encoder implementation.
- 08.05 – Bit fields. Type `Timestamp`.
- 08.06 – Benchmarks for bit fields.
- 08.07 – Fixed-size sequences of bits. Container `std::bitset`.
- 08.08 – Enumeration `std::byte`.
- 08.09 – Endianess. Big and little endian byte orders. Enumeration `std::endian`. Function `std::to_integer`.
- 08.10 – Reinterpreting bits. Operator `reinterpret_cast`. Type punning. Function `std::bit_cast`.

Long Arithmetic

- 08.11 – Indian exponentiation algorithm.
- 08.12 – Factorial for type `int`. Utility `std::numeric_limits`.
- 08.13 – Long arithmetic. Type `Integer`. Long arithmetic and comparison operators. Square root algorithm.
- 08.14 – Karatsuba fast multiplication algorithm.
- 08.15 – Appendix: main.
- 08.16 – Factorial for type `Integer`.
- 08.17 – Extended checked and unchecked integer types. Library Boost.Multiprecision.
- 08.18 – Factorial for type `boost::multiprecision::cpp_int`.
- 08.19 – Embedding Python. Python C/C++ API. Global interpreter locker. Library Boost.Python.
- 08.20 – Factorial for type `boost::python::api::object`. Module `math`.

Floating Point Types

- 08.21 – Precision. Exponent. Infinity. Quiet and signaling NaNs. Standard IEEE-754.
- 08.22 – Floating point numbers compare algorithms. Absolute and relative epsilon constants. Function `std::abs`.
- 08.23 – Extended floating point types.
- 08.24 – Numerical methods. Derivatives. Special math functions. Library Boost.Math.
- 08.25 – Weighted mean and variance. Library Boost.Accumulators.
- 08.26 – Complex numbers. Type `std::complex`. Functions `std::real`, `std::imag` and others.
- 08.27 – Discrete Fourier transform algorithm.

Random Numbers

- 08.28 – Non-deterministic generators. Entropy sources. Seeds. Engines. Distributions.
- 08.29 – Appendix: scheme.
- 08.30 – Monte-Carlo methods. Pi constant estimation.
- 08.31 – Appendix: scheme.
- 08.32 – W. L. Putnam mathematical competition problem. Probability estimation. Barycentric coordinate method.
- 08.33 – Benchmarks for branch predictor.

Chrono Management

- 08.34 – Namespace `std::chrono`. System, steady and high-resolution clocks. Time points. Unix epoch.
- 08.35 – Durations. Duration type conversions.
- 08.36 – Durations since epoch. C-style time. Type `std::time_t`. Function `std::time`.
- 08.37 – Utility `Timer`.
- 08.38 – Library Boost.Timer.
- 08.39 – Calendars. Years. Months. Days. Hours. Minutes. Seconds.
- 08.40 – Time zones.
- 08.41 – Namespace `std::literals`. User-defined literals. Literal operators.

09. Detailed Memory Management

10. Collections and Containers

11. Iterators and Algorithm Libraries

12. Text Data Processing

13. Streams and Data Serialization

14. Concurrent Programming

15. Network Technologies and Tools