# C++23 Cheat Sheet



## COMPILING

```
clang++ -std=c++23 -Wall -
Wextra \
  -o app main.cpp
```

- GCC / Clang / MSVC target C++23 via `-std=c++23`
- `<print>`/`std::println` needs GCC 14+/recent libc++/MSVC
- Fallback: `std::cout << "Hi\n";`

## PROGRAM STRUCTURE

- Translation unit = one `.cpp` + its includes
- ODR: one definition per program (inline/template exempt)
- `#pragma once` or `#ifndef` guards
- `static`/unnamed namespace = internal linkage
- Modules: `export module m;` `import m;`

## LEXICAL & LITERALS

```
0b1010'1010; 0x2A; 1'000'000
R"(raw \n string)"
5.0_km;  // user-defined
literal
[[nodiscard]] [[deprecated]]
```

## TYPES & VALUES

- `int32_t`/`uint64_t` (<cstdint>)
- `auto` deduces; `decltype` preserves refs
- `using Alias = T;` / alias templates
- Brace init `{}` rejects narrowing

## CONSTANTS & ENUMS

```
constexpr int sq(int x){return
x*x;}
consteval int cx(int x){return
x*x;}
enum class Suit { Clubs,
Hearts };
std::to_underlying(Suit::Heart
s);
using enum Suit;
```

## OPERATORS

- `a <=> b` spaceship; `= default`
- `std::cmp_less(a,b)` safe signed/unsigned
- `std::popcount`, `std::has_single_bit`

## CONTROL FLOW

```
if (auto it = m.find(k); it !=
m.end())
if constexpr (cond) { ... }
if consteval { ... } else {
... }
for (auto& [k,v] : m) { ... }
```

## FUNCTIONS & LAMBDAS

```
auto add = [](auto a, auto b)
{return a+b;};
auto fact = [](this auto self,
int n) {
  return n<=1?1:n*self(n-1);
};
```

- `noexcept`, `[[nodiscard]]`

## STRINGS & TEXT

- `std::string_view` non-owning view
- `std::format("{} {:.2f}", a, b)`
- `std::to_chars`/`from_chars`
- `"x"s`, `"x"sv` literals

## NUMBERS & MATH

- `std::numbers::pi`, `e`, `sqrt2`
- `std::mt19937 rng(seed);`
- `std::in_range<int8_t>(v)`

## CLASSES

```
class C {
  C(int x) : x_(x) {}
  int x_;
};
```

- Rule of zero > rule of five
- `static`, `friend`, `mutable`

## INHERITANCE & POLYMORPHISM

```
struct Base { virtual
~Base()=default;
  virtual double area() const
= 0; };
struct Sq : Base {
  double area() const override
{...} };
```

- Always virtual dtor when polymorphic
- `dynamic_cast`, `static_cast`, `const_cast`

## OPERATOR OVERLOADING

- Free function for asymmetric ops `(2.0*v)`
- `operator<=>` for ordering
- `operator[](r,c)` multi-arg (C++23)
- `explicit operator bool() const`

## MOVE SEMANTICS

```
T(T&& o) noexcept
  : p_(o.p_) { o.p_ = nullptr;
}
```

- `std::move` = cast to xvalue
- `std::forward<T>` preserves category

## MEMORY & SMART POINTERS

- `std::make_unique<T>(...)` — exclusive
- `std::make_shared<T>(...)` — refcounted
- `std::weak_ptr` breaks cycles
- Custom deleter: `unique_ptr<T,D>`

## TEMPLATES

```
template <typename T> T max(T
a,T b);
template <typename... Args>
auto sum(Args... a){return
(a+...);}
```

- CTAD: `Box b(42);`

## CONCEPTS

```
template<typename T>
concept Num = std::integral<T>
  || std::floating_point<T>;
T doubleIt(Num auto v){return
v*2;}
```

## COMPILE-TIME PROGRAMMING

- `static_assert(cond, "msg")`
- `std::is_integral_v<T>` traits
- `std::conditional_t`, `enable_if_t`

## NAMESPACES & PREPROCESSOR

- `inline namespace` for ABI versioning
- ADL finds free functions via arg's namespace
- `__VA_OPT__(,)` variadic macro comma

## CONTAINERS

- `vector`, `array`, `deque`, `list`
- `map`/`set` sorted, `unordered_*` O(1) avg
- `flat_map` sorted+contiguous (C++23)
- `std::span`, `std::mdspan` (C++23)

## ITERATORS & ALGORITHMS

```
std::sort(v.begin(), v.end());
std::erase_if(v, pred);
std::back_inserter(dest)
```

## RANGES & VIEWS

```
v | views::filter(even)
  | views::transform(sq)
  | views::take(3);
views::zip(a,b);
views::enumerate(a);
```

## VOCABULARY TYPES

- `std::optional<T>` + `.transform/.and_then`
- `std::variant<Ts...>` + `std::visit`
- `std::expected<T,E>` (C++23)

## CHRONO

```
auto t = steady_clock::now();
year_month_day{2024y, March,
5d};
std::format("{:%Y-%m-%d}",
date);
```

## I/O & FILESYSTEM

- `std::cin >> x;` / `getline`
- `std::ostringstream`, `ofstream(..,binary)`
- `fs::directory_iterator(dir)`
- `fs::remove(p, ec)` non-throwing

## ERROR HANDLING

```
try { risky(); }
catch (const std::exception&
e) {...}
std::error_code,
std::expected<T,E>
```

- `assert(cond)` — debug only

## THREADS

```
std::jthread t(fn, args...);
// auto-joins
std::lock_guard<std::mutex>
lk(m);
std::scoped_lock lk(m1, m2);
```

## ASYNC & ATOMICS

- `std::async(std::launch::async, fn)`
- `std::atomic<int>`, memory orders
- `std::execution::par` parallel algos

## COROUTINES

```
std::generator<int> range(int
n) {
  for (int i=0;i<n;++i)
co_yield i;
}
```

## PATTERNS

- RAII, Pimpl, NVI, CRTP
- Type erasure (`std::function`, `any`)
- Thread-safe singleton: function-local `static`

## PERFORMANCE

- Zero-overhead principle; measure first
- `const&`/`string_view`/`span` params
- `v.reserve(n)` avoids reallocation
- ASan/UBSan/TSan via `-fsanitize=...`

## BUILD & TOOLING

- CMake targets: `target_link_libraries`
- vcpkg / Conan package managers
- `clang-format`, `clang-tidy`

## TESTING

- GoogleTest: `TEST`, `TEST_F`, `EXPECT_EQ`
- Catch2: `TEST_CASE`, `SECTION`
- `ctest --output-on-failure`

## STANDARDS

- C++11 move/auto · 14 generic lambdas
- C++17 optional/filesystem · 20 concepts/ranges/coroutines
- C++23 print/expected/mdspan · C++26 reflection/contracts (not yet ISO-published)