C++ Review DUNE 💻

Una organización donde compartir notas acerca de C++ con PDFs escritos en LaTeX.

2 de mayo del 2022

📃 Pad de apuntes

📎 Liga del PDF

🎬 Sesión grabada en `diode.zone`

📅 Cronograma de esta semana

$ date (🇵🇪 Lima, 🇨🇴 Bogotá, 🇲🇽 Ciudad de México)

- `Mon May 2 07:00:00 AM -05 2022`.
- `Sun May 7 07:00:00 AM -05 2022`.

Basados en `gnuplot`

- Ejemplos
- `sciplot`
  - Ejemplos
  - Disponible en `[aur]`
- `Matplot++`
  - Ejemplos
  - Disponible en `[aur]`
- `termplotlib`
  - Ejemplos
  - Disponible en `[aur]`

Code snippet

```
cmake_minimum_required(VERSION 3.5)

project(hello-matplotlib LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Python3 COMPONENTS Interpreter Development NumPy REQUIRED)
find_package(PythonLibs 3.0 REQUIRED)

add_executable(hello-matplotlib hello-matplotlib.cc)
target_link_libraries(hello-matplotlib
    PRIVATE
        ${PYTHON_LIBRARIES}
        Python3::NumPy
)
```

Visualización

Basados en `matplotlib`

- Ejemplos
- `matplotlib-cpp`
  - Ejemplos
  - Disponible en `[aur]`

📖 `std::vector` is a sequence container that encapsulates dynamic size arrays.

Side note

If $f(x) = x^{\frac{p}{q}}$, where $\frac{p}{q}$ is a positive fraction in *lowest terms*, then you can plot as follows

- If p is even and q is odd, then graph $g(x) = |x|^{\frac{p}{q}}$ instead of $f(x)$.
- If p is odd and q is odd, then graph $g(x) = \frac{|x|}{x}|x|^{\frac{p}{q}}$ instead of $f(x)$.

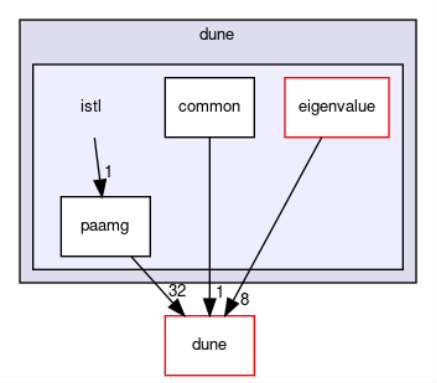
Code snippet

```cpp
#include <matplotlib-cpp/matplotlibcpp.h>

namespace plt = matplotlibcpp;

int main()
{
  std::vector<double> x = {1.5, 2.5, 3.5, 4.5},
                      y = {1, 3, 2, 4};

  plt::figure();
  plt::plot(x, y, "bo-");
  plt::xlabel("time [s]");
  plt::ylabel("observations [m]");
  plt::title("Matplotlib minimal example");
  plt::legend();
  plt::show();

  return 0;
}
```

Bibliotecas de álgebra matricial

- armadillo
  - Ejemplos
  - Disponible en `[aur]`
- Eigen
  - Ejemplos
  - Disponible en `[extra]`
- xtensor
  - Ejemplos
  - Disponible en `[aur]`
- GNU Scientific Library
  - Ejemplos
  - Disponible en `[extra]`
- `dune-matrix-vector`
  - Ejemplos
  - Disponible en `[dune-agnumpde]`
- `dune-istl`
  - Ejemplos
  - Disponible en `[aur]`
- `dune-common`
  - Ejemplos
  - Disponible en `[aur]`
- y más ...

istl Directory Reference



Standard library header `<cmath>`

Sign function

The sign function of a real number x is a piecewise function which is defined as follows

$$\operatorname{sign} x := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

📖 `std::pow` raises a number to the given power (x^y).

📖 `std::sqrt` computes square root ($\sqrt{x}$).

📖 `std::exp` returns e raised to the given power (e^x).

📖 `std::log` computes natural (base e) logarithm ($\ln x$).

📖 `std::abs` computes absolute value of an integral value ($|x|$).

📖 `std::log10` computes common (base 10) logarithm ($\log_{10} x$).

Code snippet

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

template <typename T>
int sign(T val)
{
  return (T(0) < val) - (val < T(0));
}

int main()
{
  // Primera parte
  std::cout << "5 + 3 = " << 5 + 3 << std::endl
            << "5 - 3 = " << 5 - 3 << std::endl
            << "5 * 3 = " << 5 * 3 << std::endl
            << "5 / 3 = " << 5. / 3 << std::endl
            << "⅗ = " << 3. / 5 << std::endl
            << "5³ = " << pow(5, 3) << std::endl;
```

Input/output library

Standard library header `<iomanip>`

📖 `std::setprecision` changes floating-point precision.

📖 `std::scientific` changes formatting used for floating-point I/O.

Code snippet

```
// Segunda parte
std::cout << "290 / 7 = " << std::setprecision(4 + 2)
          << 290. / 7 << std::endl;

std::cout << "290 / 7 = " << std::setprecision(15 + 2)
          << 290. / 7 << std::endl;

std::cout << "290 / 7 = " << std::scientific
          << 290. / 7 << std::endl;

std::cout << "290 / 7 = " << std::setprecision(1 + 2)
          << std::fixed << 290. / 7 << std::endl;

std::cout << "290 / 7 = " << std::setprecision(11 + 2)
          << std::fixed << 290. / 7 << std::endl;

std::cout << "290 / 7 = " << std::setprecision(0 + 2)
          << 290. / 7 << std::endl;
```

Standard library header `<cmath>`

Gamma function and the number π

- For any positive integer n, $\Gamma(n) = (n-1)!$.
- For complex numbers with a positive real part, the gamma function is defined via a convergent improper integral:

$$\Gamma(z) \coloneqq \int_0^\infty x^{z-1} e^{-x}\,\mathrm{d}x, \quad \Re(z) > 0.$$

$$\pi \coloneqq \cos^{-1}(-1).$$

📖 `std::tgamma` gamma function.

📖 `std::acos` computes arc cosine ($\arccos x$).

Code snippet

```cpp
// Tercera parte
std::cout << "√81 = " << sqrt(81) << std::endl
          << "nthroot(80, 5) = " << pow(80, 1. / 5) << std::endl
          << "e^5 = " << exp(5.) << std::endl
          << "| -24 | = " << abs(-24) << std::endl
          << "ln(1000) = " << log(1000) << std::endl
          << "log(1000) = " << log10(1000) << std::endl
          << "Γ(6) = (6 - 1)! = " << tgamma(6) << std::endl
          << "π = " << acos(-1) << std::endl
          << "sin(π / 6) = " << sin(acos(-1) / 6) << std::endl
          << "cos(π / 6) = " << cos(acos(-1) / 6) << std::endl
          << "tan(π / 6) = " << tan(acos(-1) / 6) << std::endl
          << "cot(π / 6) = " << 1 / tan(acos(-1) / 6) << std::endl;

std::cout << "round(17 / 5) = " << round(17. / 5) << std::endl
          << "fix(13 / 5) = " << trunc(13. / 5) << std::endl
          << "ceil(11 / 5) = " << ceil(11. / 5) << std::endl
          << "floor(-9 / 4) = " << floor(-9. / 4) << std::endl
          << "rem(13, 5) = " << 13 % 5 << std::endl;

std::cout << "sign(5) = " << sign<int>(5) << std::endl;
```

Utility library

📖 `std::round` rounds to nearest integer, rounding away from zero in halfway cases.
📖 `std::trunc` rounds to nearest integer not greater in magnitude than the given value.
📖 `std::ceil` computes smallest integer not less than the given value.
📖 `std::floor` computes largest integer not greater than the given value.
📖 `std::size_t` unsigned integer type returned by the sizeof operator.

Code snippet

```cpp
  // Tercera parte
  std::size_t a = 12;
  std::cout << "a = " << a << std::endl;

  std::size_t B = 4;
  std::cout << "B = " << B << std::endl;

  std::size_t C = (a - B) + 40 - a / B * 10;
  std::cout << "(a - B) + 40 - a / B * 10 = " << C << std::endl;

  std::size_t ABB = 72;
  std::cout << "ABB = " << ABB << std::endl;
  ABB = 9;
  std::cout << "ABB = " << ABB << std::endl;

  float x = 0.75;
  std::cout << "x = " << x << std::endl;

  double E = pow(sin(x), 2) + pow(cos(x), 2);
  std::cout << "sin² (x) + cos² (x) = " << E << std::endl;

  return 0;
}
```

Standard library header <complex>

📖 std::complex a complex number type.

```
template <class T>
class complex
```

📖 std::literals::complex_literals Forms a std::complex literal representing an imaginary number.

Code snippet

```
#include <iostream>
#include <iomanip>
#include <complex>
#include <cmath>

int main()
{
  using namespace std::complex_literals;
  std::cout << std::fixed << std::setprecision(1);

  std::complex<double> z1 = 1i * 1i;
  std::cout << "i² = " << z1 << "\n";

  std::complex<double> z2 = std::pow(1i, 2);
  std::cout << "i² = " << z2 << "\n";

  // no se modifica PI
  const double PI = std::acos(-1);
  std::complex<double> z3 = std::exp(1i * PI);
  std::cout << "e^iπ = " << z3 << "\n";

  std::complex<double> z4 = 1. + 2i, z5 = 1. - 2i;
  std::cout << "(1 + 2i) * (1 - 2i) = " << z4 * z5 << "\n";

  return 0;
}
```

A modern formatting library <fmt>

📖 `fmt::print`

Formats args according to specifications in fmt and writes the output to stdout.

```
template <typename... T>
void fmt::print(format_string<T...> fmt, T&&... args)
```

📖 `fmt::print`

Formats a string and prints it to stdout using ANSI escape sequences to specify text formatting.

```
template <typename S, typename... Args>
void fmt::print(const text_style &ts,
                const S &format_str,
                const Args &... args)
```

Code snippet

```
#include <cmath>
#include <fmt/core.h>
#include <fmt/color.h>

int main()
{
  // Parte a)
  double resultado_a = pow(5 - 19. / 7 + pow(2.5, 3), 2);
  fmt::print(fmt::emphasis::bold | fg(fmt::color::yellow),
             "(5 - 19 / 7 + 2.5³)² = {0:.2f}\n",
             resultado_a);

  // Parte b)
  double resultado_b = 7 * 3.1 + sqrt(120) / 5 - pow(15, 5. / 3);
  fmt::print(fmt::emphasis::bold | fg(fmt::color::green_yellow),
             "7 * 3.1 + √120 / 5 - 15 ^ (5 / 3) = {0:.2f}\n",
             resultado_b);

  // Parte c)
  double resultado_c = pow(1 / sqrt(75) + 73 / pow(3.1, 3), 1. / 4) +
                       55 * 0.41;
  fmt::print(fmt::emphasis::bold | fg(fmt::color::sky_blue),
             "pow(1 / √75 + 73 / 3.1³, ¼) + 55 * 0.41 = {0:.2f}\n",
             resultado_c);

  return 0;
}
```

Mathematical constants

Is mandatory to setup the `CMakeLists.txt` with

```
set(CMAKE_CXX_STANDARD 20)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
```

in order to use the mathematical constants

- 📖 `std::numbers::pi`
- 📖 `std::numbers::e`
- 📖 `std::numbers::egamma`
- 📖 `std::numbers::phi`

Code snippet

```cpp
#include <cmath>
#include <fmt/core.h>
#include <fmt/color.h>

int main()
{
  // constante en tiempo de compilación
  constexpr double PI = std::numbers::pi;

  double resultado_a = sin(0.2 * PI) / cos(PI / 6) + tan(PI / 180 * 72);
  fmt::print(fmt::emphasis::bold | fg(fmt::color::yellow),
             "sin(0.2π) / cos(π / 6) + tan(72°) = {0:.2f}\n",
             resultado_a);

  double resultado_b = pow(tan(PI / 180 * 64) * cos(PI / 180 * 15), 2) +
                       pow(sin(PI / 180 * 37) / cos(PI / 180 * 20), 2);
  fmt::print(fmt::emphasis::bold | fg(fmt::color::orange),
             "(tan(64°) * cos(15°))² + (sin(37°) / cos(20°))² = {0:.2f}\n",
             resultado_b);

  return 0;
}
```

A modern formatting library

$$\text{expr}_1(x, y) = \left(x^2 + y^2\right)^{\frac{2}{3}} + \frac{xy}{y - x}$$

$$\text{expr}_2(x, y) = \frac{\sqrt{x + y}}{(x - y)^2} + 2x^2 - xy^2$$

Code snippet

```
#include <cmath>
#include <fmt/core.h>
#include <fmt/color.h>

int main()
{
  float x = 6.5, y = 3.8;
  auto expr1 = [](float x, float y)
  {
    return pow(pow(x, 2) + pow(y, 2), 2. / 3) +
           (x * y) / (y - x);
  };
  double resultado_a = expr1(x, y);
  fmt::print(fmt::emphasis::bold | fg(fmt::color::light_green),
             "(x² + y²) ^ ⅔ + (x * y) / (y - x) = {0:.2f}\n",
             resultado_a);

  auto expr2 = [](float x, float y)
  {
    return sqrt(x + y) / pow(x - y, 2) + 2 * pow(x, 2) -
           x * pow(y, 2);
  };
  double resultado_b = expr2(x, y);
  fmt::print(fmt::emphasis::bold | fg(fmt::color::blue_violet),
             "√(x + y) / (x - y)² + 2 * x² - x * y² = {0:.2f}\n",
             resultado_b);

  return 0;
}
```

Unit testing framework for C++

<catch2>

📖 TEST_CASE

`TEST_CASE( test name [, tags ] )`

📖 REQUIRE

`REQUIRE( expression )`

📖 Approx

Code snippet

```
#define CATCH_CONFIG_MAIN

#include <catch2/catch.hpp>

#include "student.hh"

TEST_CASE("Comparing values from first")
{
  REQUIRE(result_1a == Approx(320.79f));
  REQUIRE(result_1b == Approx(-67.342f));
  REQUIRE(result_1c == Approx(23.816f).margin(.1f));
}

TEST_CASE("Comparing values from second")
{
  REQUIRE(result_2a == Approx(3.7564f));
  REQUIRE(result_2b == Approx(4.3323f));
}

TEST_CASE("Comparing values from third")
{
  REQUIRE(expr1(x, y) == Approx(5.6091f));
  REQUIRE(expr2(x, y) == Approx(-8.9198f));
}
```

Actividad `cmath-example` en `src/student.hh`.

Code snippet

```cpp
#pragma once // prevents multiple definitions

// Test for exercise 1
double result_1a = 0; // TODO: Complete
double result_1b = 0; // TODO: Complete
double result_1c = 0; // TODO: Complete

// Test for exercise 2
double result_2a = 0; // TODO: Complete
double result_2b = 0; // TODO: Complete

// Test for exercise 3
float x = 6.5, y = 3.8;

auto expr1 = [](float x, float y)
{
  return 0; // TODO: Complete
};

auto expr2 = [](float x, float y)
{
  return 0; // TODO: Complete
};
```