C++ Review DUNE 💻

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

8 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`.

CMake as a build system

Documentación de CMake

- cmake_minimum_required
- project
- find_package
- list
- include
- add_executable
- target_link_libraries

Documentación de Dune build system

- dune_project
- dune_enable_all_packages
- finalize_dune_project
- target_link_dune_default_libraries

```cmake
macro(target_link_dune_default_libraries _target)
  foreach(_lib ${DUNE_LIBS})
    target_link_libraries(${_target} PUBLIC ${_lib})
  endforeach()
endmacro(target_link_dune_default_libraries)
```

Code snippet

```cmake
cmake_minimum_required(VERSION 3.13)
project(dune-vector-learn CXX)

#find dune-common and set the module path
find_package(dune-common REQUIRED)
list(APPEND CMAKE_MODULE_PATH ${dune-common_MODULE_PATH})

#include the dune macros
include(DuneMacros)

# start a dune project with information from dune.module
dune_project()

dune_enable_all_packages()

add_executable("dune-vector-learn" dune-vector-learn.cc)
target_link_dune_default_libraries("dune-vector-learn")

add_executable("identity" identity.cc)
target_link_dune_default_libraries("identity")

# finalize the dune project, e.g. generating config.h etc.
finalize_dune_project(GENERATE_CONFIG_H_CMAKE)
```

ArrayList

📖 Dune::TestSuite

A Simple helper class to organize your test suite.

```
TestSuite(std::string name = "",
          ThrowPolicy policy = ThrowOnRequired)
```

📖 Dune::ArrayList A dynamically growing random access list.

```
template <class T,
          int N = 100,
          class A = std::allocator<T>>
class Dune::ArrayList<T, N, A>
```

Code snippet

```
#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include <iostream>
#include <dune/common/test/testsuite.hh>
#include <dune/common/arraylist.hh>

int main(int argc, char **argv)
{
  Dune::TestSuite test;
  Dune::ArrayList<double, 10> alist;

  for (int i = 0; i < 100; i++)
    alist.push_back(i);

  for (auto &e : alist)
    std::cout << e << std::endl;

  return test.exit();
}
```

FieldVector

📖 [[maybe_unused]] Suppresses warnings on unused entities.

📖 Dune::FieldVector vector space out of a tensor product of fields.

```
template <class K, int SIZE>
class Dune::FieldVector<K, SIZE>
```

📖 Dune::printvector Print an ISTL vector.

```
void Dune::printvector(std::ostream &s,
                       const V &v,
                       std::string title,
                       std::string rowtext,
                       int columns = 1,
                       int width = 10,
                       int precision = 2)
```

📖 DUNE_THROW Macro to throw an exception.

```
#define DUNE_THROW (E, m)
```

📖 Dune::Exception Base class for Dune-Exceptions.

Code snippet

```
#include <dune/common/fvector.hh>
#include <dune/istl/io.hh>
#include <iostream>

int main()
{
  [[maybe_unused]] int p = 0;
  constexpr int dim = 3;
  Dune::FieldVector<double, dim> x(0);
  Dune::printvector(std::cout, x, "x", "row");

  try
  {
    if (x.dimension ≠ 2)
      DUNE_THROW(Dune::Exception,
                 "DUNE_ASSERT_AND_RETURN returned incorrect dimension")
  }
  catch (Dune::Exception &e)
  {
    std::cerr << "Dune reported error: " << e << std::endl;
    return 1;
  }
  catch ( ... )
  {
    std::cerr << "Unknown exception thrown!" << std::endl;
    return 1;
  }
}
```

std::vector

$v \in \mathbb{Z}^{100}$.

Code snippet

```cpp
#include <fmt/ranges.h>
#include <iostream>
#include <vector>

int main()
{
  std::vector<int> v{};

  for (int i = 0; i < 100; i++)
    v.push_back(i);

  fmt::print("Primera forma sin iterador:\n{}\n", v);

  for (int i = 0; i < v.size(); i++)
    v[i] *= 2;

  fmt::print("Segunda forma con iterador:\n");

  std::vector<int>::iterator iter = v.begin();
  while (iter != v.end())
  {
    std::cout << *iter << std::endl;
    iter++;
  }
}
```

Matrix

📖 Dune::Matrix A generic dynamic dense matrix.

```
template <class T, class A = std::allocator<T>>
class Dune::Matrix<T, A>
```

📖 Dune::printmatrix Print a generic block matrix.

```
void Dune::printmatrix(std::ostream &s,
                       const M &A,
                       std::string title,
                       std::string rowtext,
                       int width = 10,
                       int precision = 2)
```

Die Maximumsnorm, Tschebyschew-Norm oder ∞-Norm (Unendlich-Norm) eines Vektors ist definiert als

$$\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$$

Die euklidische Norm oder 2-Norm eines Vektors ist definiert als

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

Die Summennorm, (genauer) Betragssummennorm, oder 1-Norm (lies: „Einsnorm“) eines Vektors ist definiert als

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

Code snippet

```
#include <dune/common/fmatrix.hh>
#include <dune/istl/io.hh>
#include <dune/istl/matrix.hh>
#include <fmt/core.h>
#include <fmt/ranges.h>

int main()
{
  constexpr int dim = 2;
  Dune::FieldVector<double, dim> x(0);
  Dune::printvector(std::cout, x, "x", "row");

  fmt::print("x = {}\n", x);
  fmt::print("Size of x: {}\n", x.size());
  fmt::print("‖x‖₂ = {}\n", x.two_norm());
  fmt::print("‖x‖∞: {}\n", x.infinity_norm());
  fmt::print("Dimension of x: {}\n", x.dimension);

  Dune::Matrix<double> matrix(3, 5);
  matrix = 0;

  Dune::printmatrix(std::cout, matrix, "Matrix<double>", "--");
  fmt::print("matrix has {} rows.\n", matrix.N());
  fmt::print("matrix has {} columns.\n", matrix.M());

  return 0;
}
```

Polynomial

Die Frobeniusnorm $\|\cdot\|_F$ einer Komplexe Zahl $(m \times n)$-Matrix $A \in \mathbb{K}^{m \times n}$ mit $\mathbb{K}$ aus dem Körper der reellen oder komplexen Zahlen ist definiert als

$$\|A\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}.$$

📖 `Dune::Functions::Polynomial` A scalar polynomial implementation.

```
template <class K>
class Dune::Functions::Polynomial<K>
```

Code snippet

```
#include <iostream>

#include <dune/functions/analyticfunctions/polynomial.hh>

int main(int argc, char **argv)
{
  auto p = Dune::Functions::Polynomial<int>({1, 2, 3});
  auto a_i = p.coefficients();

  std::cout << "P[x] = "
            << a_i[0] << "* x ^ " << 0 << " + "
            << a_i[1] << "* x ^ " << 1 << " + "
            << a_i[2] << "* x ^ " << 2 << "\n"
            << "P[x = 0]: " << p(0) << "\n"
            << "P[x = 1]: " << p(1) << "\n"
            << "P[x = 2]: " << p(2) << "\n";

  return 0;
}
```

range

Code snippet

```cpp
#include <fmt/ranges.h>
#include <iostream>
#include <vector>

std::vector<double> range(double min, double max, std::size_t N)
{
  std::vector<double> range;
  double delta = (max - min) / double(N - 1);
  for (int i = 0; i < N; i++)
  {
    range.push_back(min + i * delta);
  }
  return range;
}
```

range

Code snippet

```cpp
int main()
{
  using MyDVector = std::vector<double>;

  std::vector<int> v1 = {1, 2, 3, 4};
  fmt::print("{}\n", v1);

  std::vector<int> v2;
  v2 = std::vector<int>(v1.begin() + 1, v1.end() - 1);
  fmt::print("{}\n", v2);

  MyDVector u1 = range(1, 2, 100);
  fmt::print("{}\n", u1);

  MyDVector u2 = range(1., 21., 20);
  fmt::print("{}\n", u2);

  return 0;
}
```