4-constants

June 3, 2024

1 Physical and mathematical constants

1.1 Physical constants

7 constants were fixed in 2019. Each of them is used to define one of the 7 units of the SI : * Frequency of the hyperthin Cesium transition, 9192631770 Hz, defines second, * The speed of light in space, 299792458 m/s, defines meter, * Planck constant, 6.62607015×10−34 J s, defines kilogram, * Elementary charge, 1.602176634×10−19 C, defines ampere, * Boltzmann constant, 1.380649×10−23 J/K, defines kelvin, * Avogadro constant, 6.02214076×1023 mol−1, defines mole, * Energetic intensity of 540 THz radiation, 683 lm/W, defines candela.

[1]:
```
%%file tmp.constants.cpp

#include <iostream>
#include "phys/units/io.hpp"
#include "phys/units/physical_constants.hpp"

namespace pu = phys::units;
using namespace phys::units::io;

int main()
 {
  std::cout << std::endl ;
  std::cout.precision(16) ;
  std::cout << "speed of light in a vacuum: " << pu::c << std::endl ;
  std::cout << "Planck constant: " << pu::h << std::endl ;
  std::cout << "elementary charge: " << pu::e << std::endl ;
  std::cout << "Avogadro constant: " << pu::N_sub_A << std::endl ;
  std::cout << std::endl ;
  std::cout << "acceleration of free-fall: " << pu::g_sub_n << std::endl ;
  std::cout << "electronvolt: " << pu::eV << std::endl ;
  std::cout << "unified atomic mass unit: " << pu::u << std::endl ;
  std::cout << std::endl ;
 }
```

```
Writing tmp.constants.cpp
```

[2]:
```
!rm -f tmp.constants.exe && g++ -I. -std=c++17 tmp.constants.cpp -o tmp.
↪constants.exe
```

[3]:
```
!./tmp.constants.exe
```

```
speed of light in a vacuum: 299792458 m/s
Planck constant: 6.62606876e-34 m+2 kg s-1
elementary charge: 1.602176462e-19 C
Avogadro constant: 6.02214199e+23 mol-1

acceleration of free-fall: 9.806649999999999 m s-2
electronvolt: 1.60217733e-19 J
unified atomic mass unit: 1.6605402e-27 kg
```

1.2 Mathematical constants

We are slighty getting out of the strict SI system, but some mathematical constants, such as PI, are intensively used, and the problematic of their definition is close to the physical constants : is it enough to defined them in the most precise available floating point type ?

1.2.1 PI

Nowadays, for what concerns PI, experts dislike predefined literal constants (`M_PI`, or `M_PIl` with gcc, or some BOOST constant), and prefer use some variable template with the target precision, and a compile time computation based on `atan(1) * 4` or `acos(-1)` :

[4]:
```
%%file tmp.constants.cpp

#include <iostream>
#include <cmath>

template<typename T>
T constexpr pi = std::acos(static_cast<T>(-1)) ;

int main()
 {
  // 3.14159 26535 89793 23846 26433
  std::cout.precision(20) ;
  std::cout<< pi<int> <<"\n";
  std::cout<< pi<float> <<"\n";
  std::cout<< pi<double> <<"\n";
  std::cout<< pi<long double> <<"\n";
 }
```

```
Overwriting tmp.constants.cpp
```

[5]:
```
!rm -f tmp.constants.exe && g++ -I. -std=c++17 tmp.constants.cpp -o tmp.
↪constants.exe
```

[6]:
```
!./tmp.constants.exe
```

```
3
3.1415927410125732422
3.141592653589793116
3.1415926535897932385
```

BEWARE: for those who might be tempt to use `long double` to have a better precision, you have to remember `long double` size depend on the plateform, so this type is not portable…

This code is supposed to be independent from both the plateform and the most precise type of the moment. The debate is still raging to know which is the best computation formula, and which are the compilers doing it really at compilation time (especially the call to `acos` or `atan`). **This leads some authors to consider `constexpr` as non-portable**.

C++20 provides values in the standard library. The flavor without the `_v` prefix is the shortcut for the double version.

[7]:
```
%%file tmp.constants.cpp

#include <iostream>
#include <numbers>

int main()
 {
  std::cout.precision(20) ;
  std::cout << std::endl ;
  std::cout << "pi (float)       : " << std::numbers::pi_v<float> <<"\n";
  std::cout << "pi (double)      : " << std::numbers::pi_v<double> <<"\n";
  std::cout << "pi (long double) : " << std::numbers::pi_v<long double> <<"\n";
  std::cout << std::endl ;
  std::cout.precision(16) ;
  std::cout << "pi  : " << std::numbers::pi <<"\n";
  std::cout << "e   : " << std::numbers::e <<"\n";
  std::cout << "phi : " << std::numbers::phi <<"\n";
  std::cout << std::endl ;
 }
```

```
Overwriting tmp.constants.cpp
```

[8]:
```
!rm -f tmp.constants.exe && g++ -I. -std=c++20 tmp.constants.cpp -o tmp.
 ↪constants.exe
```

[9]:
```
!./tmp.constants.exe
```

```
pi (float)       : 3.1415927410125732422
pi (double)      : 3.141592653589793116
pi (long double) : 3.1415926535897932385

pi  : 3.141592653589793
e   : 2.718281828459045
```

```
phi : 1.618033988749895
```

2 Questions ?

2.1 Ressources & inspirations

- https://www.meetingcpp.com/blog/items/cpp-and-pi.html
- https://stackoverflow.com/questions/49778240/does-c11-14-17-or-20-introduce-a-standard-constant-for-pi
- https://stackoverflow.com/questions/1727881/how-to-use-the-pi-constant-in-c
- https://stackoverflow.com/questions/18773343/how-to-calculate-euler-constant-or-euler-powered-in-c/57285506#57285506
- http://wg21.link/p0631
- https://en.cppreference.com/w/cpp/numeric/constants