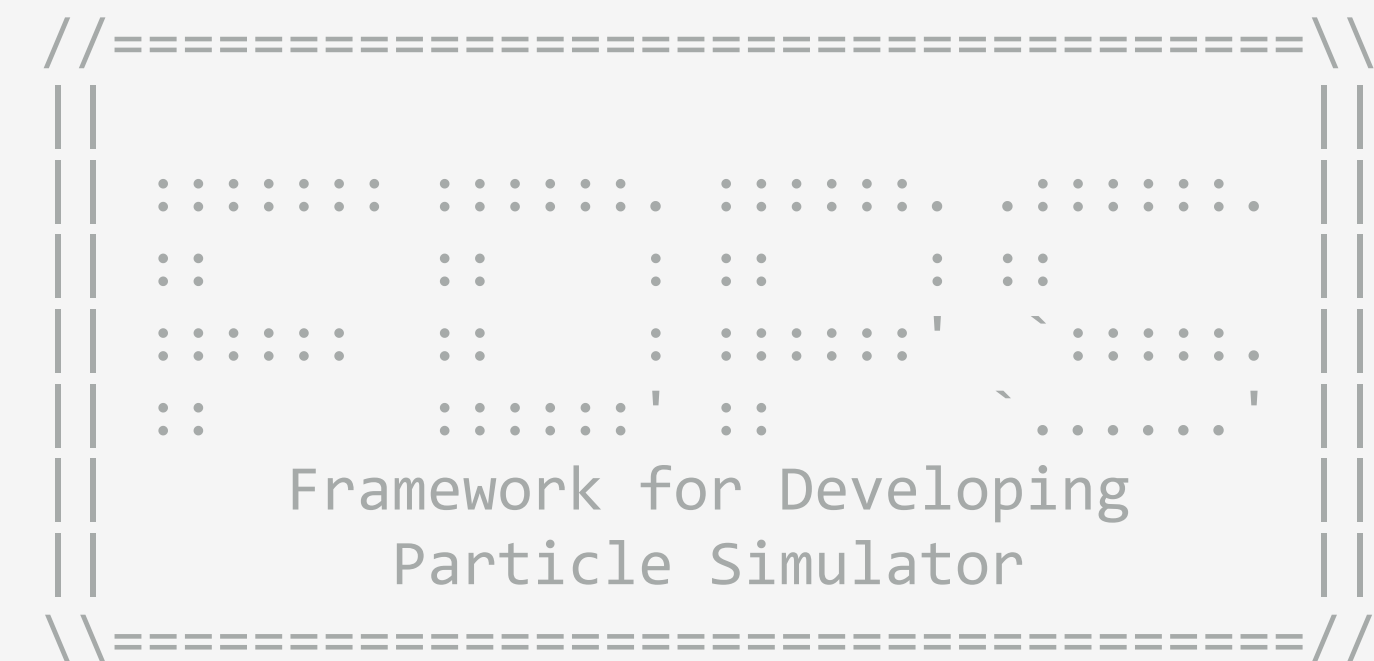


FDPS 講習会 C++編

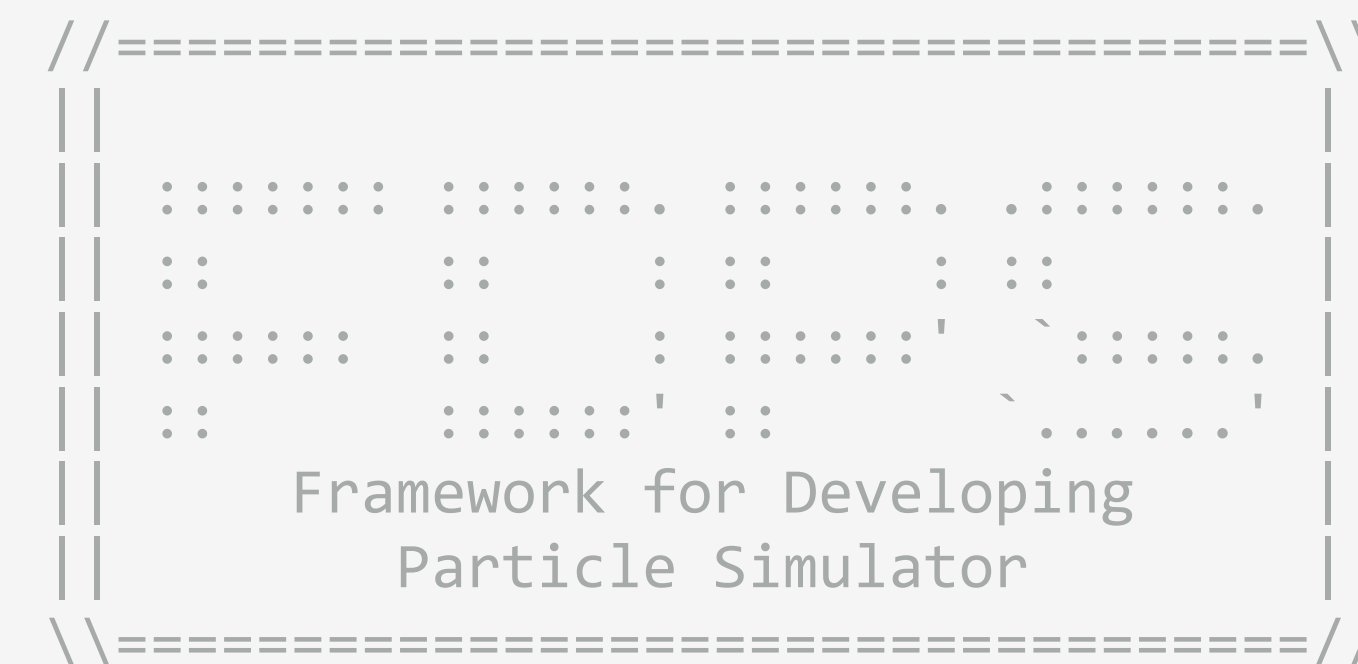
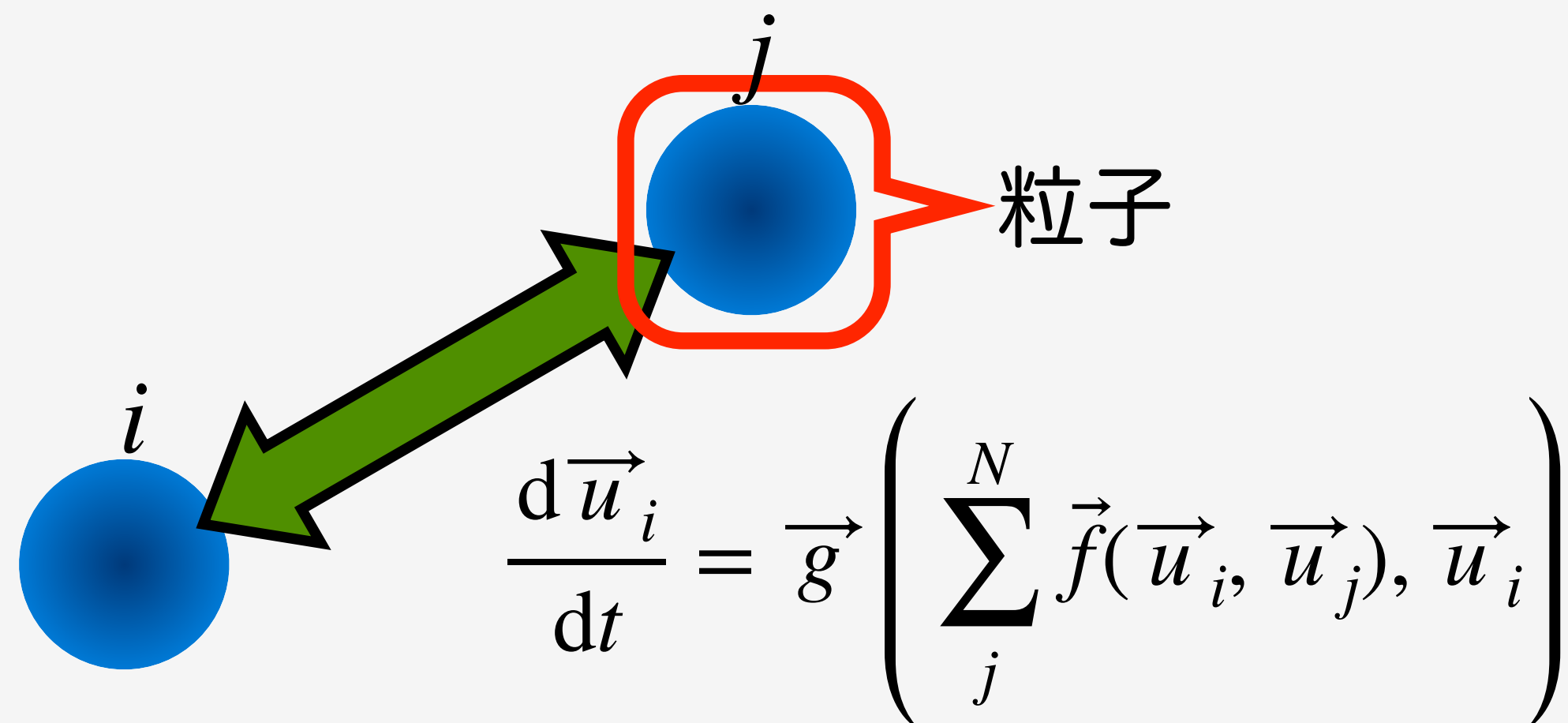
細野 七月 (神戸大学/RIKEN CCS/JAMSTEC)

2021/09/09 FDPS講習会



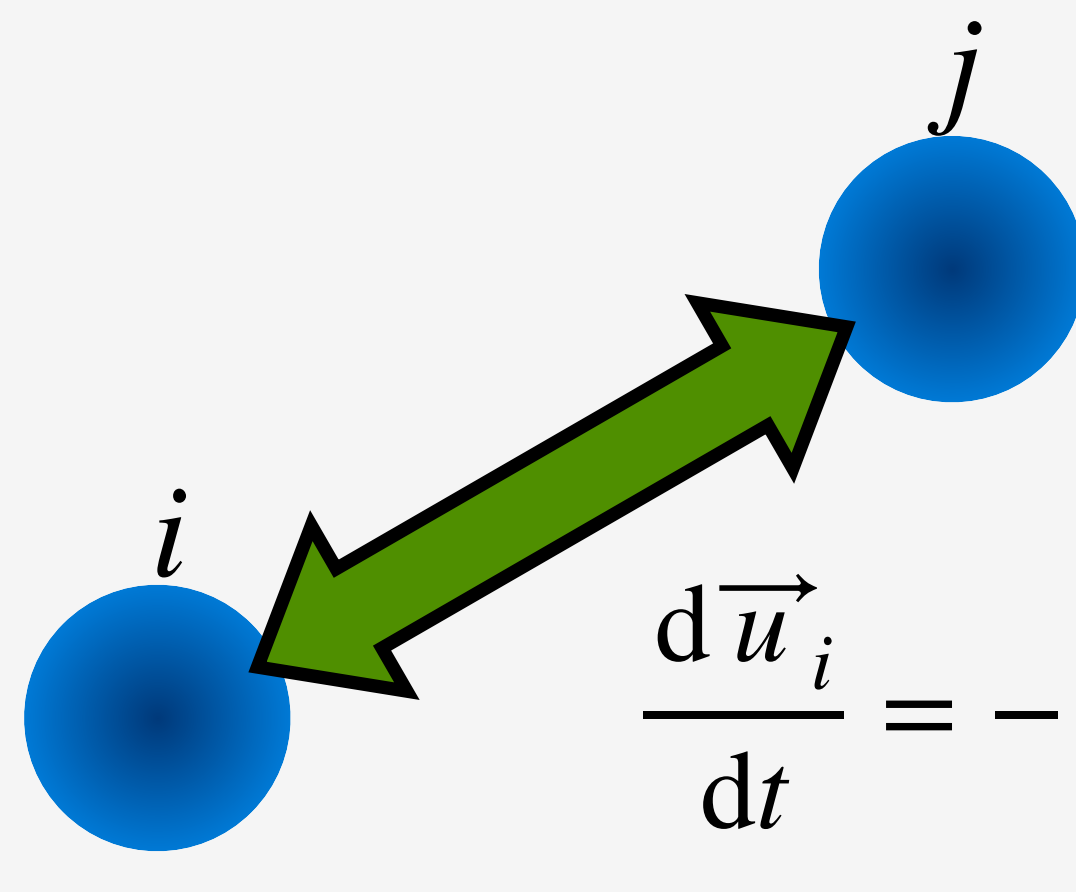
FDPSが扱うもの

◆個々の要素(粒子)に作用する力が、**粒子間相互作用の重ね合わせ**で記述できるもの

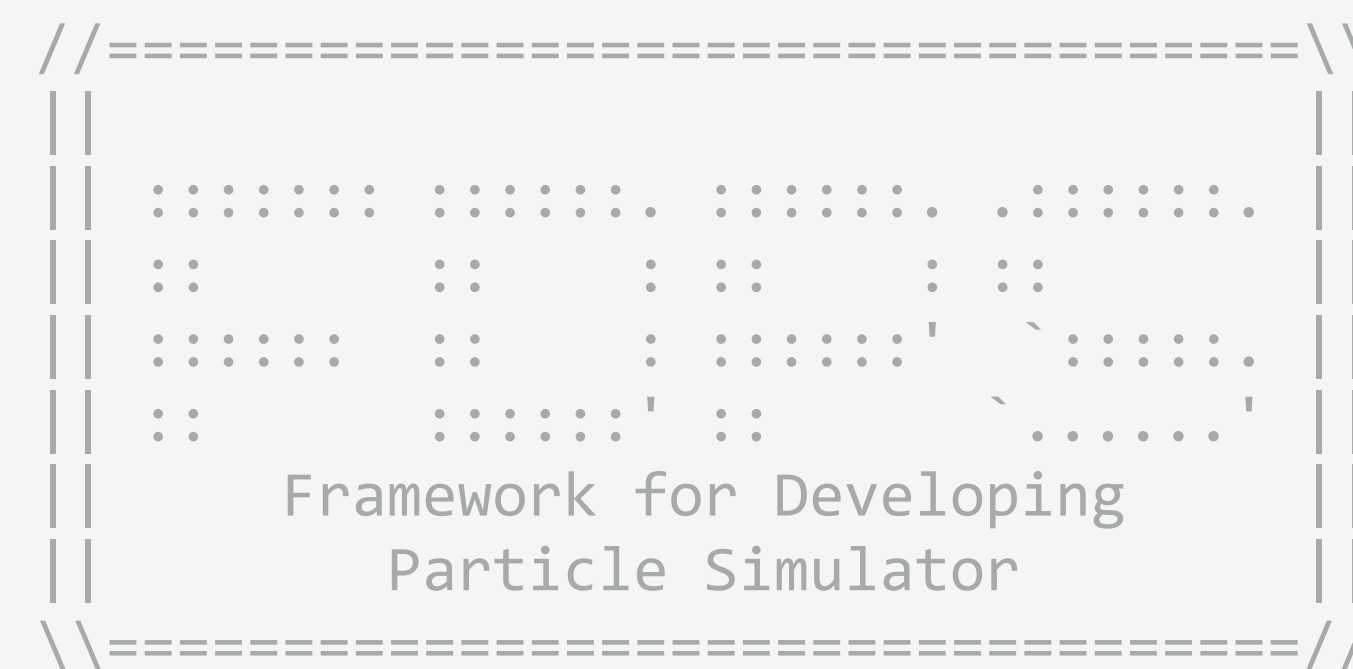


FDPSが扱うもの

- ◆個々の要素(粒子)に作用する力が、粒子間相互作用の重ね合わせで記述できるもの


$$\frac{d\vec{u}_i}{dt} = - \sum_j^N \frac{m_j}{|\vec{x}_i - \vec{x}_j|^3} (\vec{x}_i - \vec{x}_j)$$

例：ニュートン重力



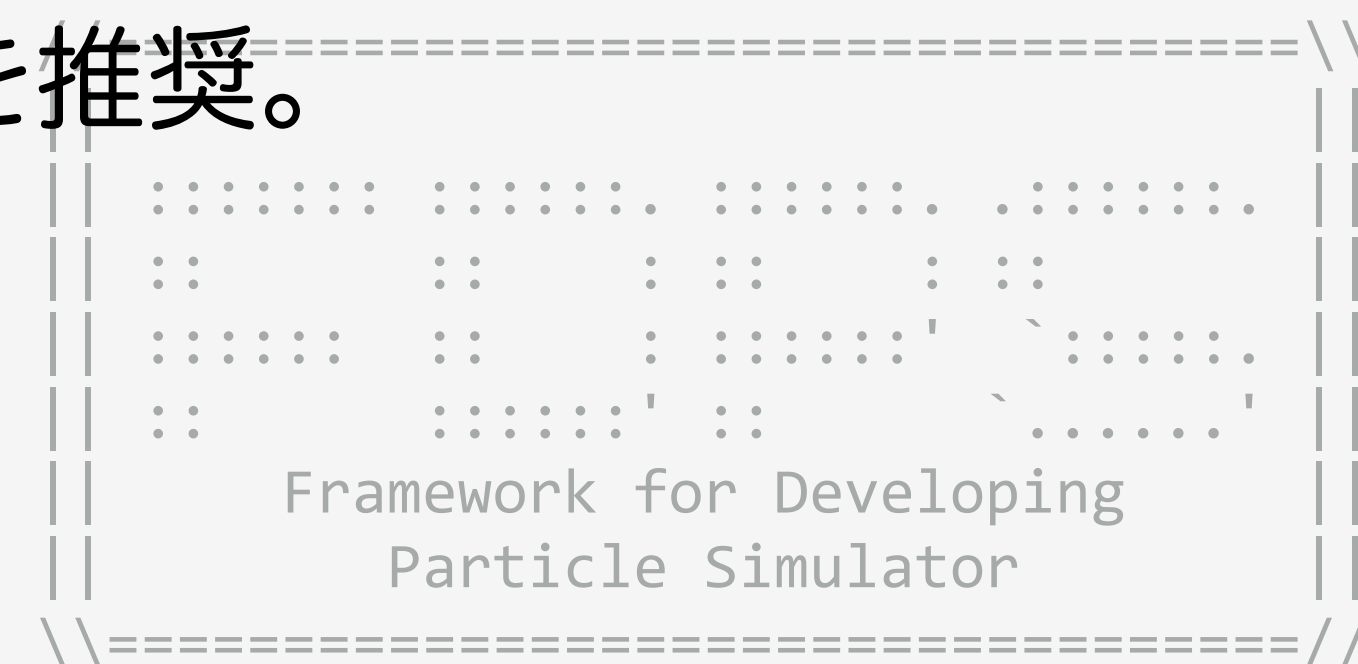
C++の機能：クラスとは

◆複数のデータをまとめたもの
Cで言うところの構造体

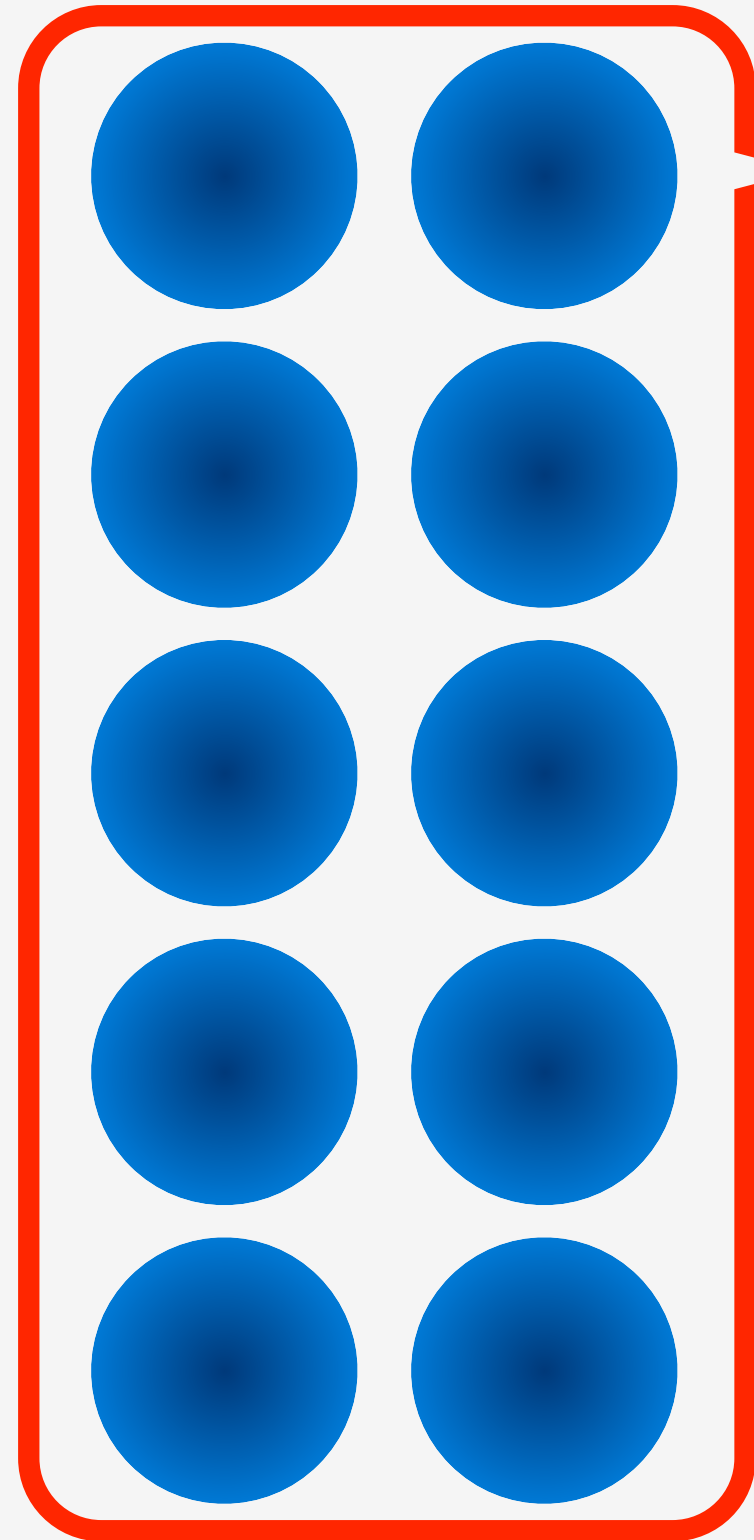
◆**クラス**を使うと

- 演算子を定義出来る。
- サブルーチンに値を送るときに楽。
- 後からデータ付け足すのも楽。

◆FDPSでは、ユーザーは粒子データをまとめた**粒子クラス**を記述する必要がある。
また、2D/3Dベクトルをやりとりするのに便利な**ベクトルクラス**が用意されている。
位置や速度にはこのFDPSが用意した**ベクトルクラス**を用いることを推奨。



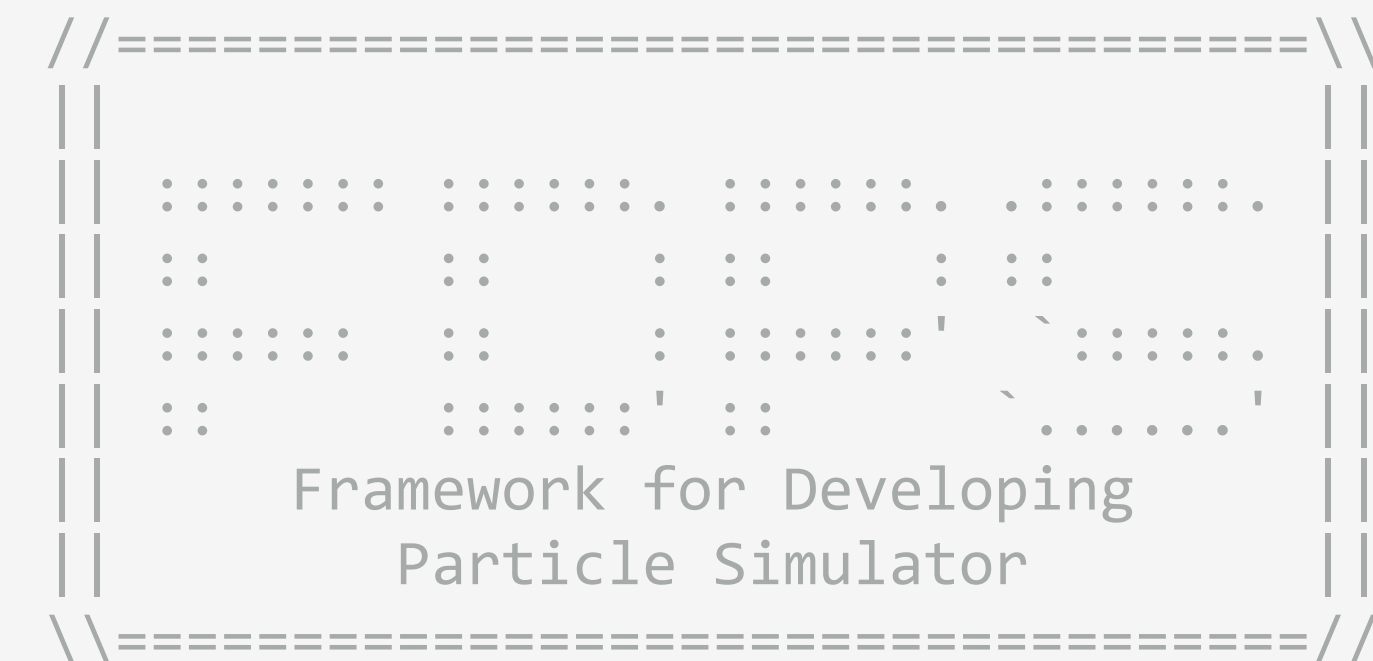
C++の機能：クラス例(粒子クラス)



Nbody個の粒子

必要なデータは…

- mass
- position(x, y, z)
- velocity(x, y, z)
- acceleration(x, y, z)



C++の機能：クラスのメリット

- ◆ **クラス**を使うとサブルーチンに値を送るときに簡単に書ける。
後からデータを付け足すのも楽。

```
void doSomething(double mass[],
                double position[][3],
                double velocity[][3],
                double acceleration[][3]){
    //do something here
}

void doSomething(FPGrav particle[]){
    //do something here
}
```

```
//=====\\
||
|| .....
||  ::  ::  ::  ::
|| .....
||
|| .....
||  ::  ::  ::  ::
|| .....
||
|| Framework for Developing
|| Particle Simulator
||
||=====\\
```


C++の機能：メンバ関数

◆ **クラス**内の変数(メンバ変数)達に対して、何らかの処理を加えたい時に記述するもの。
アクセスには.演算子を用いる。

◆ FDPSの場合、
FDPSとユーザーコード間でデータのやりとりをするための **メンバ関数**を書く必要がある。

```
class FPGrav{
    PS::F64    mass;
    PS::F64vec position;
    PS::F64vec velocity;
    PS::F64vec acceleration;
    PS::F64vec getPos(){
        return position;
    }
}body[Nbody];

PS::F64vec position = body[0].getPos();
```

```
//=====\\
|
|  ::::::::::::::::::::::::::::::::::::
|  ::      ::      ::      ::      ::
|  ::      ::      ::      ::      ::
|  ::      ::      ::      ::      ::
|  ::      ::      ::      ::      ::
|
|  Framework for Developing
|  Particle Simulator
|
|=====\\
```


サンプルコード used_defined.hpp

```
    PS::F64    r3_inv = rij * rij + eps2;
    PS::F64    r_inv  = 1.0/sqrt(r3_inv);
    r3_inv     = r_inv * r_inv;
    r_inv      *= ep_j[j].getCharge();
    r3_inv     *= r_inv;
    ai         -= r3_inv * rij;
    poti       -= r_inv;
  }
  force[i].acc += ai;
  force[i].pot += poti;
}
}

#endif
```

```
//=====\\
||
|| .....  .....  .....  .....
||  ::      ::      :  :      :
|| .....  ::      :  .....  .....
||  ::      .....  :  .....  .....
||
|| Framework for Developing
|| Particle Simulator
||
//=====\\
```


サンプルコード nbody.cpp

```
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<sys/stat.h>
#include<particle_simulator.hpp>
#ifdef ENABLE_PHANTOM_GRAPE_X86
#include <gp5util.h>
#endif
#ifdef ENABLE_GPU_CUDA
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include "user-defined.hpp"

void makeColdUniformSphere(const PS::F64 mass_glb,
                           const PS::S64 n_glb,
                           const PS::S64 n_loc,
                           PS::F64 *& mass,
                           PS::F64vec *& pos,
                           PS::F64vec *& vel,
                           const PS::F64 eng = -0.25,
                           const PS::S32 seed = 0) {

    assert(eng < 0.0);
    {
        PS::MTTS mt;
        mt.init_genrand(0);
    }
}
```

```
//=====\\
||
||  :::::::::: :::::::::: :::::::::: ::::::::::
||  ::      ::      : :      : :
||  :::::::::: : :      : :      : :
||  ::      :::::::::: : :      : :
||
||  Framework for Developing
||  Particle Simulator
||
||=====\\
```

サンプルコード nbody.cpp

```
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<sys/stat.h>
#include<particle_simulator.hpp>
#ifdef ENABLE_PHANTOM_GRAPE_X86
#include <gp5util.h>
#endif
#ifdef ENABLE_GPU_CUDA
#define MULTI_WALK
#include"force_gpu_cuda.hpp"
#endif
#include "user-defined.hpp"
```

FDPSのヘッダー読み込み

```
void makeColdUniformSphere(const PS::F64 mass_glb,
                           const PS::S64 n_glb,
                           const PS::S64 n_loc,
                           PS::F64 *& mass,
                           PS::F64vec *& pos,
                           PS::F64vec *& vel,
                           const PS::F64 eng = -0.25,
                           const PS::S32 seed = 0) {
```

```
    assert(eng < 0.0);
    {
```

```
        PS::MTTS mt;
        mt.init_genrand(0);
```

```
//=====\\
||
||  :::::::::: :::::::::: :::::::::: ::::::::::
||  ::      ::      : :      : :
||  :::::::::: : :      : :      : :
||  ::      :::::::::: : :      : :
||
||  Framework for Developing
||  Particle Simulator
||
||=====\\
```

サンプルコード nbody.cpp

```
}
```

```
template<class Tpsys>
void kick(Tpsys & system,
         const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].vel += system[i].acc * dt;
    }
}
```

粒子数が取得可能

```
template<class Tpsys>
void drift(Tpsys & system,
          const PS::F64 dt) {
    PS::S32 n = system.getNumberOfParticleLocal();
    for(PS::S32 i = 0; i < n; i++) {
        system[i].pos += system[i].vel * dt;
    }
}
```

```
template<class Tpsys>
void calcEnergy(const Tpsys & system,
               PS::F64 & etot,
               PS::F64 & ekin,
               PS::F64 & epot,
               const bool clear=true){
    if(clear){
```

```
//=====\\
||
||  ::::::::::: ::::::::::: ::::::::::: :::::::::::
||  ::          ::          : :          : :
||  ::::::::::: ::          : :          : :
||  ::          ::::::::::: ' :          '
||
||          Framework for Developing
||          Particle Simulator
||
||=====\\
```


サンプルコード nbody.cpp

```
    free_grav.calcGravity<CalcGravity<PS::SPJMonopole>,
    CalcGravity<PS::SPJMonopole>,
    system_grav,
    dinfo);

#endif
    kick(system_grav, dt * 0.5);
    n_loop++;
}
#ifdef ENABLE_PHANTOM_GRAPE_X86
    g5_close();
#endif
    PS::Finalize();
    return 0;
}
```

FDPS終了

```
//=====\\
||
|| .....
||  ::  ::  ::  ::
|| .....  ::  ::  ::
||  ::  ::  ::  ::
|| .....
||
|| Framework for Developing
|| Particle Simulator
||
||=====\\
```