C++について
&
サンプルコード解説

細野七月、岩澤全規

習得しておきたいC++の機能

- FDPS用いるにおいて、知っておきたいC++の機能
  - 名前空間
    - PS::F64 など
  - クラス
    - メンバ変数とメンバ関数
  - テンプレート
    - template <typename T> など
  - 標準ライブラリ
    - std::vectorやstd::cout など
    - ただし今回のサンプルコードではI/O用のstd::coutなどが使われている程度なので省略

これ以上の事は基本的に必要ない。

C++の機能：名前空間とは

- グローバルな変数、関数、型名などの名前被りを防ぐ機能。
  FDPSでは、F64vecといった型が存在している(後述)が、これはFDPSが用意したPSという名前空間に含まれているので、ユーザーが独自にF64vecという型を定義する事ができる。

- 名前空間へのアクセスは::演算子を用いる。

```
PS::F64vec v_FDPS; // FDPS組み込みベクトル型
F64vec v_user; // ユーザー定義ベクトル型
```

C++の機能：クラスとは

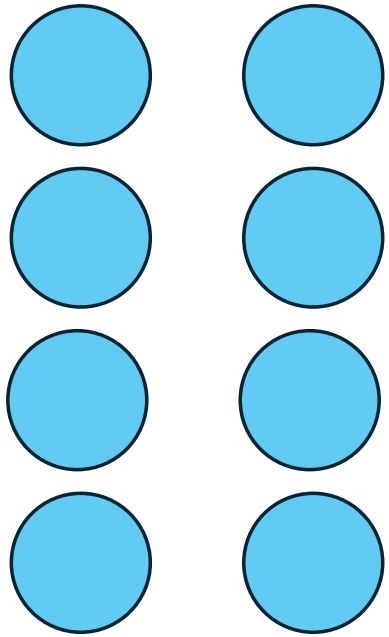
- 複数のデータをまとめたもの。Cで言うところの構造体。
- クラスを使うと
  - 演算子を定義出来る。
  - サブルーチンに値を送るときに楽。
  - 後からデータ付け足すのも楽。
- FDPSでは、ユーザーは粒子データをまとめた粒子クラスを記述する必要がある。
  また、2D/3Dベクトルをやりとりするのに便利なベクトルクラスが用意されている。
  位置や速度にはこのFDPSが用意したベクトルクラスを用いることを

C++の機能：クラス例(ベクトルクラス)

- 数学演算子が既に定義済みのため、直感的に書ける。
  C++のI/Oである、std::coutにわたす事も可能。

```
PS::F64vec v1 = PS::F64vec( 1.0, 2.5, -3.0);
PS::F64vec v2 = PS::F64vec(-2.0, -1.5, 2.0);
PS::F64vec v_add = v1 + v2;
PS::F64vec v_sub = v1 - v2;
PS::F64 inner = v1 * v2;
PS::F64vec outer = v1 ^ v2;
std::cout << v_add.x << std::endl;
std::cout << v_sub << std::endl;
```

C++の機能：クラス例(粒子クラス)

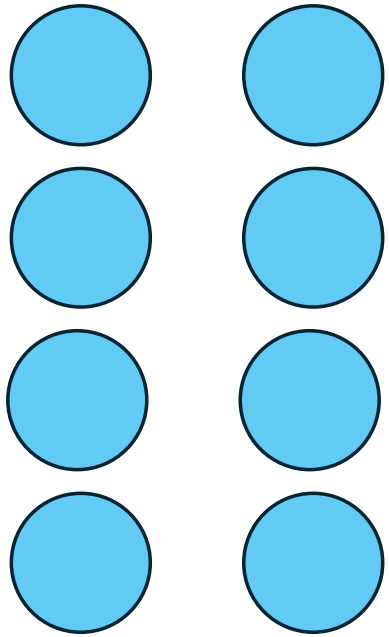


N個の粒子

必要なデータは

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

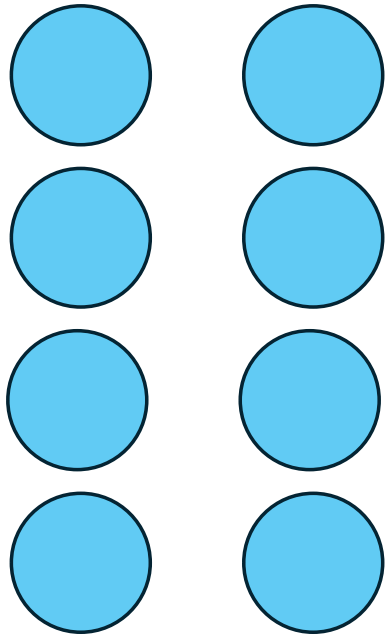
C++の機能：クラス例(粒子クラス)



N個の粒子

```
double mass[N];
double position[N][3];
double velocity[N][3];
double acceleration[N][3]
```

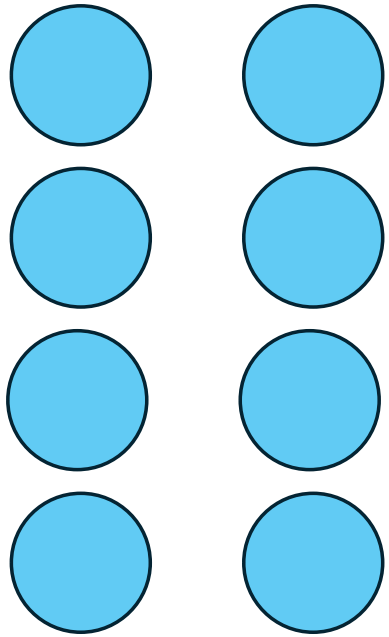
C++の機能：クラス例(粒子クラス)



N個の粒子

```
class FPGrav{
  double mass;
  double position[3];
  double velocity[3];
  double acceleration[3];
} body[N];
```

C++の機能：クラス例(粒子クラス)



N個の粒子

```
class FPGrav{
  PS::F64 mass;
  PS::F64vec position;
  PS::F64vec velocity;
  PS::F64vec acceleration;
} body[N];
//クラス内クラスも可能
```

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
}
```

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();
```

C++の機能：テンプレート

- 同じ処理を違う型に対して行いたい時に書くもの。
  クラスをテンプレート化したクラステンプレートと、
  関数をテンプレート化した関数テンプレートが存在する。
- まずクラステンプレートについて解説し、その後関数テンプレートに関して解説する。

C++の機能：クラステンプレート

- FDPSでは、粒子群クラステンプレートの＜＞の中に粒子クラスを入れる事で、粒子群クラスの実体が作られる。同様に、相互作用ツリークラスの実体も作られる。

```
//粒子群クラス
PS::ParticleSystem<FPGrav> system_grav;
//相互作用ツリークラス
PS::TreeForForceLong<FPGrav, FPGrav, FPGrav> Monopole tree_grav;
```

```
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();
```

C++の機能：関数テンプレート

- FDPSでは相互作用関数は関数テンプレートを用いて記述出来る。
- 関数テンプレートを用いずに記述する事も可能だが、相互作用が長距離力の場合、
  計算内容的には全く同じものにも関わらず、
  近傍の粒子からの相互作用とツリーノードからの相互作用2つを書かなければならない。
- 関数テンプレートを用いると記述するのは1つだけで済むので、今回は関数テンプレートを用いたサンプルコードを紹介する。

C++の機能：関数テンプレート(続き)

- FDPSでは、ツリーノードクラスは既に用意されている(PS::SPJMonopole)ため、それを使う。
- 相互作用関数はSPJMonopoleも使うので、getPos();などを用いて書く必要がある。

```
template <class TparticleJ>
void CalcGravity(const FPGrav * ep_i,
                 const PS::S32 n_ip,
                 const TParitcleJ * ep_j,
                 const PS::S32 n_jp,
                 FPGrav * force){
  for(PS::S32 i = 0 ; i < n_ip ; i ++){
    PS::F64vec xi = ep_i[i].getPos();
    for(PS::S32 j = 0 ; j < n_jp ; j ++){
      PS::F64vec rij = x_i - ep_j[j].getPos();
      //calc gravity
    }
  }
}
```

コード構成

- ユーザーが書くべきものは、
  - #include <particle_simulator.h>
  - 粒子クラスと必要なメンバ関数
  - 相互作用関数
  - 時間積分ルーチン
  - I/O (粒子クラスのI/Oと、FileHeaderクラス)

サンプルコード

- 今回のサンプルコードの内容
  - 粒子間相互作用は、重力
    - Phantom-GRAPE (Tanikawa+, 2011; 2012) やPIKGにも対応
  - 時間積分法はleap-frog法
  - 初期条件はその場生成
    - ファイル読み込みではない
- ファイル構成
  - user-defined.hpp
  - nbody.cpp

  他にも色々入っているが必要なものはすべてこの2つの中にある

user-defined.hpp

```
#pragma once
class FileHeader{
public:
    PS::S64 n_body;
    PS::F64 time;
    PS::S32 readAscii(FILE * fp) {
        if(fscanf(fp, "%lf\n", &time) != 1) exit(1);
        if(fscanf(fp, "%ld\n", &n_body) != 1) exit(1);
        return n_body;
    }
    void writeAscii(FILE* fp) const {
        fprintf(fp, "%e\n", time);
        fprintf(fp, "%ld\n", n_body);
    }
};
```

粒子クラス

```
class FPGrav{
public:
    PS::S64    id;
    PS::F64    mass;
    PS::F64vec pos;
    PS::F64vec vel;
    PS::F64vec acc;
    PS::F64    pot;
    static PS::F64 eps;

    PS::F64vec getPos() const {
        return pos;
    }

    PS::F64 getCharge() const {
        return mass;
    }

    void copyFromFP(const FPGrav & fp){
        mass = fp.mass;
        pos  = fp.pos;
    }

    void copyFromForce(const FPGrav & force) {
        acc = force.acc;
        pot = force.pot;
    }

    void clear() {
        acc = 0.0;
        pot = 0.0;
    }

    void writeAscii(FILE* fp) const {
        fprintf(fp, "%ld\t%g\t%g\t%g\t%g\t%g\t%g\t%g\n",
        this->id, this->mass,
        this->pos.x, this->pos.y, this->pos.z,
        this->vel.x, this->vel.y, this->vel.z);
    }

    void readAscii(FILE* fp) {
        if(fscanf(fp, "%ld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
          &this->id, &this->mass,
          &this->pos.x, &this->pos.y, &this->pos.z,
          &this->vel.x, &this->vel.y, &this->vel.z) != 8) exit(-1);
        }
};
```

物理量

必須メンバ関数

位置座標を返す

質量(電荷)を返す

FPから相互作用に

必要な物理量をEP

ForceからFPに値を

Forceクラスのデータ

```
#ifdef ENABLE_PHANTOM_GRAPE_X86
```

相互作用関数(phantom grape)

```
template <class TParticleJ>
void CalcGravity(const FPGrav * iptcl,
                 const PS::S32 ni,
                 const TParticleJ * jptcl,
                 const PS::S32 nj,
                 FPGrav * force) {
    const PS::S32 nipipe = ni;
    const PS::S32 njpipe = nj;
    PS::F64 (*xi)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);
    PS::F64 (*ai)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * nipipe * PS::DIMENSION);
    PS::F64  *pi     = (PS::F64  *    )malloc(sizeof(PS::F64) * nipipe);
    PS::F64 (*xj)[3] = (PS::F64 (*)[3])malloc(sizeof(PS::F64) * njpipe * PS::DIMENSION);
    PS::F64  *mj     = (PS::F64  *    )malloc(sizeof(PS::F64) * njpipe);
    for(PS::S32 i = 0; i < ni; i++) {
        xi[i][0] = iptcl[i].getPos()[0];
        xi[i][1] = iptcl[i].getPos()[1];
        xi[i][2] = iptcl[i].getPos()[2];
        ai[i][0] = 0.0;
        ai[i][1] = 0.0;
        ai[i][2] = 0.0;
        pi[i]    = 0.0;
    }
    for(PS::S32 j = 0; j < nj; j++) {
        xj[j][0] = jptcl[j].getPos()[0];
        xj[j][1] = jptcl[j].getPos()[1];
        xj[j][2] = jptcl[j].getPos()[2];
        mj[j]    = jptcl[j].getCharge();
        xj[j][0] = jptcl[j].pos[0];
        xj[j][1] = jptcl[j].pos[1];
        xj[j][2] = jptcl[j].pos[2];
        mj[j]    = jptcl[j].mass;
    }
    PS::S32 devid = PS::Comm::getThreadNum();
    g5_set_xmjMC(devid, 0, nj, xj, mj);
    g5_set_nMC(devid, nj);
    g5_calculate_force_on_xMC(devid, xi, ai, pi, ni);
    for(PS::S32 i = 0; i < ni; i++) {
        force[i].acc[0] += ai[i][0];
        force[i].acc[1] += ai[i][1];
        force[i].acc[2] += ai[i][2];
        force[i].pot    -= pi[i];
    }
    free(xi);
    free(ai);
    free(pi);
    free(xj);
    free(mj);
}

#elif USE_PIKG_KERNEL
struct Epi{
  PS::F32vec pos;
};
struct Epj{
  PS::F32vec pos;
  PS::F32    mass;
};
struct Force{
  PS::F32vec acc;
  PS::F32    pot;
};



#include "kernel_pikg.hpp"

```

```
template <class TParticleJ>
void CalcGravity(const FPGrav * ep_i,
                 const PS::S32 n_ip,
                 const TParticleJ * ep_j,
                 const PS::S32 n_jp,
                 FPGrav * force) {
  Epi epi[n_ip];
  Force f[n_ip];
  for(int i=0;i<n_ip;i++){
    epi[i].pos = (PS::F32vec)(ep_i[i].pos - ep_i[0].pos);

    f[i].acc = force[i].acc;
    f[i].pot = force[i].pot;
  }
  Epj epj[n_jp];
  for(int i=0;i<n_jp;i++){
    epj[i].pos = (PS::F32vec)(ep_j[i].pos - ep_i[0].pos);
    epj[i].mass = ep_j[i].mass;
  }
  CalcGravityEpEp(FPGrav::eps*FPGrav::eps)(epi,n_ip,epj,n_jp,f);
  for(int i=0;i<n_ip;i++){
    force[i].acc = f[i].acc;
    force[i].pot = f[i].pot;
  }
}
```

相互作用関数(PIKG)

```
#else
template <class TParticleJ>
void CalcGravity(const FPGrav * ep_i,
                 const PS::S32 n_ip,
                 const TParticleJ * ep_j,
                 const PS::S32 n_jp,
                 FPGrav * force) {
    PS::F64 eps2 = FPGrav::eps * FPGrav::eps;
    for(PS::S32 i = 0; i < n_ip; i++){
        PS::F64vec xi = ep_i[i].getPos();
        PS::F64vec ai = 0.0;
        PS::F64 poti = 0.0;
        for(PS::S32 j = 0; j < n_jp; j++){
            PS::F64vec rij    = xi - ep_j[j].getPos();
            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
```

関数テンプレート

相互作用関数

メンバ関数呼び出し

nbody.cpp

```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);
        for (PS::S32 i = 0; i < n_loc; i++) {
            mass[i] = mass_glb / n_glb;
            const PS::F64 radius = 3.0;
            do {
                pos[i][0] = (2. * mt.genrand_res53() - 1.) * radius;
                pos[i][1] = (2. * mt.genrand_res53() - 1.) * radius;
                pos[i][2] = (2. * mt.genrand_res53() - 1.) * radius;
            } while (pos[i] * pos[i] >= radius * radius);
            vel[i][0] = 0.0;
            vel[i][1] = 0.0;
            vel[i][2] = 0.0;
        }
    }

    PS::F64vec cm_pos = 0.0;
    PS::F64vec cm_vel = 0.0;
    PS::F64 cm_mass = 0.0;
    for (PS::S32 i = 0; i < n_loc; i++) {
        cm_pos += mass[i] * pos[i];
        cm_vel += mass[i] * vel[i];
        cm_mass += mass[i];
    }
    cm_pos /= cm_mass;
    cm_vel /= cm_mass;
    for (PS::S32 i = 0; i < n_loc; i++) {
        pos[i] -= cm_pos;
        vel[i] -= cm_vel;
    }
}

template <class Tpsys>
void setParticlesColdUniformSphere(Tpsys &psys, const PS::S32 n_glb, PS::S32 &n_loc) {
    n_loc = n_glb;
    psys.setNumberOfParticleLocal(n_loc);

    PS::F64 *mass = new PS::F64[n_loc];
    PS::F64vec *pos = new PS::F64vec[n_loc];
    PS::F64vec *vel = new PS::F64vec[n_loc];
    const PS::F64 m_tot = 1.0;
    const PS::F64 eng = -0.25;
    makeColdUniformSphere(m_tot, n_glb, n_loc, mass, pos, vel, eng);
    for (PS::S32 i = 0; i < n_loc; i++) {
        psys[i].mass = mass[i];
        psys[i].pos = pos[i];
```

FDPSヘッダーの読み込み

```
65          psys[i].vel = vel[i];
66          psys[i].id = i;
67      }
68      delete[] mass;
69      delete[] pos;
70      delete[] vel;
71  }
72
73  template <class Tpsys>
74  void kick(Tpsys &system, const PS::F64 dt) {
75      PS::S32 n = system.getNumberOfParticleLocal();自プロセスが担当する粒子数を返す
76      for (PS::S32 i = 0; i < n; i++) {
77          system[i].vel += system[i].acc * dt;
78      }
79  }                                  粒子群クラスに[i]をつけると配列の様に粒子データへアクセス可能
80
81  template <class Tpsys>
82  void drift(Tpsys &system, const PS::F64 dt) {
83      PS::S32 n = system.getNumberOfParticleLocal();
84      for (PS::S32 i = 0; i < n; i++) {
85          system[i].pos += system[i].vel * dt;
86      }
87  }
88
89  template <class Tpsys>
90  void calcEnergy(const Tpsys &system, PS::F64 &etot, PS::F64 &ekin, PS::F64 &epot, const bool clear = true) {
91      if (clear) {
92          etot = ekin = epot = 0.0;
93      }
94      PS::F64 etot_loc = 0.0;
95      PS::F64 ekin_loc = 0.0;
96      PS::F64 epot_loc = 0.0;
97      const PS::S32 nbody = system.getNumberOfParticleLocal();
98      for (PS::S32 i = 0; i < nbody; i++) {
99          ekin_loc += system[i].mass * system[i].vel * system[i].vel;
100         epot_loc += system[i].mass * (system[i].pot + system[i].mass / FPGrav::eps);
101     }
102     ekin_loc *= 0.5;
103     epot_loc *= 0.5;
104     etot_loc = ekin_loc + epot_loc;
105     etot = PS::Comm::getSum(etot_loc);
106     epot = PS::Comm::getSum(epot_loc);
107     ekin = PS::Comm::getSum(ekin_loc);
108 }
109
110 void printHelp() {
111     std::cerr << "o: dir name of output (default: ./result)" << std::endl;
112     std::cerr << "t: theta (default: 0.5)" << std::endl;
113     std::cerr << "T: time_end (default: 10.0)" << std::endl;
114     std::cerr << "s: time_step (default: 1.0 / 128.0)" << std::endl;
115     std::cerr << "d: dt_diag (default: 1.0 / 8.0)" << std::endl;
116     std::cerr << "D: dt_snap (default: 1.0)" << std::endl;
117     std::cerr << "l: n_leaf_limit (default: 8)" << std::endl;
118     std::cerr << "n: n_group_limit (default: 64)" << std::endl;
119     std::cerr << "N: n_tot (default: 1024)" << std::endl;
120     std::cerr << "h: help" << std::endl;
121 }
122
123 void makeOutputDirectory(char *dir_name) {
124     struct stat st;
125     PS::S32 ret = 0;
126     if (PS::Comm::getRank() == 0) {
127         if (stat(dir_name, &st) != 0) {
128             ret = mkdir(dir_name, 0777);
129         } else {
130             ret = 0;  // the directory named dir_name already exists.
131         }
```

```
    }
    PS::Comm::broadcast(&ret, 1);
    if (ret == 0) {
        if (PS::Comm::getRank() == 0) fprintf(stderr, "Directory \"%s\" is successfully made.\n", dir_name);
    } else {
        if (PS::Comm::getRank() == 0) fprintf(stderr, "Directory %s fails to be made.\n", dir_name);
        PS::Abort();
    }
}

PS::F64 FPGrav::eps = 1.0 / 32.0;

int main(int argc, char *argv[]) {
    std::cout << std::setprecision(15);
    std::cerr << std::setprecision(15);

    PS::Initialize(argc, argv);
    PS::F64 theta = 0.5;
    PS::S32 n_leaf_limit = 8;
    PS::S32 n_group_limit = 64;
    PS::F64 time_end = 10.0;
    PS::F64 dt = 1.0 / 128.0;
    PS::F64 dt_diag = 1.0 / 8.0;
    PS::F64 dt_snap = 1.0;
    //PS::S32 n_steps_reuse = 4;
    PS::S32 n_steps_reuse = 1;
    bool clear_force = true;
    char dir_name[1024];
    PS::S64 n_tot = 1024;
    PS::S32 c;
    sprintf(dir_name, "./result");
    opterr = 0;
    while ((c = getopt(argc, argv, "i:o:d:D:t:T:l:n:N:hs:")) != -1) {
        switch (c) {
            case 'o':
                strncat(dir_name, optarg, 1000);
                break;
            case 't':
                theta = atof(optarg);
                std::cerr << "theta =" << theta << std::endl;
                break;
            case 'T':
                time_end = atof(optarg);
                std::cerr << "time_end = " << time_end << std::endl;
                break;
            case 's':
                dt = atof(optarg);
                std::cerr << "time_step = " << dt << std::endl;
                break;
            case 'd':
                dt_diag = atof(optarg);
                std::cerr << "dt_diag = " << dt_diag << std::endl;
                break;
            case 'D':
                dt_snap = atof(optarg);
                std::cerr << "dt_snap = " << dt_snap << std::endl;
                break;
            case 'l':
                n_leaf_limit = atoi(optarg);
                std::cerr << "n_leaf_limit = " << n_leaf_limit << std::endl;
                break;
            case 'n':
                n_group_limit = atoi(optarg);
                std::cerr << "n_group_limit = " << n_group_limit << std::endl;
                break;
            case 'N':
                n_tot = atoi(optarg);
```

メイン関数は引数を取る

FDPSの初期化

```
                std::cerr << "n_tot = " << n_tot << std::endl;
                break;
            case 'h':
                if (PS::Comm::getRank() == 0) {
                    printHelp();
                }
                PS::Finalize();
                return 0;
            default:
                if (PS::Comm::getRank() == 0) {
                    std::cerr << "No such option! Available options are here." << std::endl;
                    printHelp();
                }
                PS::Abort();
        }
    }

    makeOutputDirectory(dir_name);

    std::ofstream fout_eng;

    if (PS::Comm::getRank() == 0) {
        char sout_de[1280];
        sprintf(sout_de, "%s/t-de.dat", dir_name);
        fout_eng.open(sout_de);
        fprintf(stdout, "This is a sample program of N-body simulation on FDPS!\n");
        fprintf(stdout, "Number of processes: %d\n", PS::Comm::getNumberOfProc());
        fprintf(stdout, "Number of threads per process: %d\n", PS::Comm::getNumberOfThread());
    }

    PS::ParticleSystem<FPGrav> system_grav;
    system_grav.initialize();
    PS::S32 n_loc = 0;
    PS::F32 time_sys = 0.0;
    if (PS::Comm::getRank() == 0) {
        setParticlesColdUniformSphere(system_grav, n_tot, n_loc);
    } else {
        system_grav.setNumberOfParticleLocal(n_loc);
    }

    const PS::F32 coef_ema = 0.3;
    PS::DomainInfo dinfo;
    dinfo.initialize(coef_ema);
    dinfo.decomposeDomainAll(system_grav);
    system_grav.exchangeParticle(dinfo);
    n_loc = system_grav.getNumberOfParticleLocal();

#if defined(ENABLE_PHANTOM_GRAPE_X86)
    g5_open();
    g5_set_eps_to_all(FPGrav::eps);
#endif

    PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;
    tree_grav.initialize(n_loc, theta, n_leaf_limit, n_group_limit);
    //tree_grav.let_usage_mode = PS::LET_USAGE_MODE::CONSTRUCT_GLOBAL_TREE;
    tree_grav.let_usage_mode_ = PS::LET_USAGE_MODE::CONSTRUCT_LET_TREE;

    /*
    // CHECKED
    tree_grav.setParticleLocalTree(system_grav);
    tree_grav.makeLocalTree(dinfo);
    tree_grav.makeGlobalTree(dinfo);
    tree_grav.calcMomentGlobalTree();
    tree_grav.calcForce(CalcGravity<FPGrav>, CalcGravity<PS::SPJMonopole>);
    for(int i=0; i<system_grav.getNumberOfParticleLocal(); i++){
    system_grav[i].copyFromForce(tree_grav.getForce(i));
    std::cout<<"i= "<<i<<" acc= "<<system_grav[i].acc<<std::endl;
```

粒子群クラスの生成と初期化

ドメイン情報クラスの生成と初期化

粒子交換

自プロセスが担当する粒子数を返すメンバ関数

相互作用ツリークラスの生成と初期化

```
    }
    PS::Finalize();
    return 0;
    // CHECKED
    */

#if defined(MULTI_WALK_PTCL)
    const PS::S32 n_walk_limit = 4;
    const PS::S32 tag_max = 1;
    tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP, RetrieveKernel, tag_max, system_grav, dinfo, n_walk_limit);
#elif defined(ENABLE_GPU_CUDA)
    const PS::S32 n_walk_limit = 4;
    const PS::S32 tag_max = 1;
    tree_grav.calcForceAllAndWriteBackMultiWalkIndex(DispatchKernelWithSPIndex, RetrieveKernel, tag_max, system_grav, dinfo, n_walk_limit);
#else
    tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>, CalcGravity<PS::SPJMonopole>, system_grav, dinfo);
#endif
    PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
    calcEnergy(system_grav, Etot0, Ekin0, Epot0);
    PS::F64 time_diag = 0.0;
    PS::F64 time_snap = 0.0;
    PS::S64 n_loop = 0;
    PS::S32 id_snap = 0;
    while (time_sys <= time_end) {
    /*
    if(PS::Comm::getRank()==0){
        if(n_loop % 16 == 0){
        ((system_grav.getTimeProfile() + dinfo.getTimeProfile() + tree_grav.getTimeProfile())/16).dump(1);
        system_grav.clearTimeProfile();
        dinfo.clearTimeProfile();
        tree_grav.clearTimeProfile();
        }
    }
    */

        if ((time_sys >= time_snap) || ((time_sys + dt) - time_snap) > (time_snap - time_sys)) {
            char filename[1280];
            sprintf(filename, "%s/%04d.dat", dir_name, id_snap++);
            FileHeader header;
            header.time = time_sys;
            header.n_body = system_grav.getNumberOfParticleGlobal();
            system_grav.writeParticleAscii(filename, header);
            time_snap += dt_snap;
        }

        calcEnergy(system_grav, Etot1, Ekin1, Epot1);

        if (PS::Comm::getRank() == 0) {
            if ((time_sys >= time_diag) || ((time_sys + dt) - time_diag) > (time_diag - time_sys)) {
                fout_eng << time_sys << "   " << (Etot1 - Etot0) / Etot0 << std::endl;
                fprintf(stdout, "time: %10.7f energy error: %+e\n", time_sys, (Etot1 - Etot0) / Etot0);
                time_diag += dt_diag;
            }
        }

        kick(system_grav, dt * 0.5);
        time_sys += dt;
        drift(system_grav, dt);
        auto list_mode = PS::REUSE_LIST;
        if (n_loop % n_steps_reuse == 0) {
            dinfo.decomposeDomainAll(system_grav);
            system_grav.exchangeParticle(dinfo);
            list_mode = PS::MAKE_LIST_FOR_REUSE;
```

相互作用関数を用いた力の計算

引数は順にEP-EPの相互作用関数、EP-SPの相互作用関数、

積分のメインループ開始

velocity kick
速度を半ステップ進める

drift
位置を1ステップ進める

```
331         }
332 #if defined(MULTI_WALK_PTCL)
333     // multi walk
334         tree_grav.calcForceAllAndWriteBackMultiWalk(DispatchKernelWithSP, RetrieveKernel, tag_max, system_grav,
    dinfo, n_walk_limit, clear_force, list_mode);
335 #elif defined(ENABLE_GPU_CUDA)
336     // multi walk + index
337         tree_grav.calcForceAllAndWriteBackMultiWalkIndex(DispatchKernelWithSPIndex, RetrieveKernel, tag_max,
    system_grav, dinfo, n_walk_limit, clear_force, list_mode);
338 #else
339         tree_grav.calcForceAllAndWriteBack(CalcGravity<FPGrav>, CalcGravity<PS::SPJMonopole>, system_grav, dinfo,
    clear_force, list_mode);
340 #endif
341
342     /*
343     if(PS::Comm::getRank() == 0){
344         (dinfo.getTimeProfile() + system_grav.getTimeProfile() + tree_grav.getTimeProfile()).dump(2);
345         std::cout<<"***************"<<std::endl;
346     }
347     dinfo.clearTimeProfile();
348     system_grav.clearTimeProfile();
349     tree_grav.clearTimeProfile();
350     */
351         kick(system_grav, dt * 0.5);
352         n_loop++;
353     }
354
355 #if defined(ENABLE_PHANTOM_GRAPE_X86)
356     g5_close();
357 #endif
358
359     PS::Finalize();
360     return 0;
361 }
362
```

velocity kick
速度を半ステップ進める

FDPSの終了処理