FDPSユーザーのための C++

FDPS講習会2018

似鳥啓吾

はじめに

- 世界の計算機センターでサポートされているプログラミング言語
  - 事実上C++とFortranの二択
- FDPSはというと
  - C++のヘッダライブラリとして実装されている
    - 後述する「テンプレート」という機能を多様しているため
- ユーザーコード
  - FDPSのヘッダを#includeしてC++で書く（ネイティブ）
  - Fortran 2003で書いてトランスコーダを使う（隣の教室）
- とりあえずC言語の経験者ぐらいを対象に、FDPSのユーザーコード開発に必要となるC++の機能を紹介していきます

言語のバージョンについて

- 言語にも「バージョン」があります
  - FORTRAN 66/77, Fortran 90/95/2003/2008
  - C++ 98/03/11/14/17/20
- FDPS FortranではFortran 2003の新仕様を使っています
- FDPS C++では「京」での利用を優先してC++03までにとどめてきました
  - 最近コンパイラの対応が改善したのでC++11/14は解禁となりました
  - ユーザーコードでもC++11/14を使ってよいと思います
  - （年号は言語仕様がfixされた年なのでコンパイラや判例が出揃うまでには年月がかかります）

習得しておきたいC++の機能

- 名前空間
  - 「PS::」や「std::」
- クラス（構造体） with メンバ関数
  - 「継承」や「仮想関数」といったオブジェクト指向機能はFDPSを使う分には不要
- テンプレート
  - 用意されたものを使うことができれば十分
  - FDPS提供はクラステンプレートを提供、ユーザーはこれを実体化
  - メタプログラミングとかはユーザー側では不要
- 標準ライブラリSTL
  - とりあえずstd::vectorを可変長配列として使えるぐらいで十分
  - あとは簡単なIO（C言語のものを使ってもいい）

名前空間（namespace）

- グローバルな変数名、関数名、型名などの衝突を避けるための仕組
  - ディレクトリ（フォルダ）みたいなもの
  - C言語にはなかった「`::`」という演算子でアクセスする
- FDPSで提供される関数や型は`PS`という名前空間に
- C++の標準ライブラリは`std`という名前空間に
  - `std::cout`や`std::endl`など

- ユーザープログラムで新規に名前空間を定義する必要はありません
- サンプルコードに沢山出現する`PS::`や`std::`に驚かないぐらいの理解で大丈夫です

クラス（構造体）

- C言語の構造体
  - ```
    struct Vector3{
        double x, y, z;
    };
    ```
  - データをパックして新しい型を作る
  - 複数の型を混在させることもできる
- C++での拡張
  - ```
    struct Vector3{
        double x, y, z;
        double norm() const {
            return sqrt(x*x + y*y + z*z);
        }
    };
    ```
  - 「メンバ変数」に加えて「メンバ関数」も書けるようになった
  - C++での`class`と`struct`は機能としては同じもの

クラス（構造体）の使用例

```
struct F64vec{
    double x, y, z;

    const F64vec &operator+=(const F64vec &rhs){
        x += rhs.x; y += rhs.y; z += rhs.z;
        return (*this);
    }
    friend F64vec operator*(const double s, const F64vec &v){
        F64vec3 t = {s*v.x, s*v.y, s*v.z};
        return t;
    }
};

struct Particle{
    PS::F64vec pos, vel, acc;

    void kick(const double dt){
        vel += dt * acc;
    }
    void drift(const double dt){
        pos += dt * vel;
    }
};
void integrate(Particle &p, const double dt){
    p.kick (dt);
    p.drift(dt);
}
```

空間ベクトル型と演算子の定義

ここはFDPS側が提供

ユーザー定義の粒子構造体

メンバ関数で数値積分

メンバ関数を呼び出してみる

テンプレート (template)

- クラスまたは関数の「雛形」
- 「クラステンプレート」と「関数テンプレート」とある
  - *テンプレート名 <型名>*
    のように<>で「テンプレート引数」を渡すことで、「実体化」したクラスや関数が得られる。
- FDPSでは以下のように変数宣言する
  - `PS::ParticleSystem<FPGrav> system_grav;`
  - `PS::TreeForForceLong<FPGrav, FPGrav, FPGrav>::Monopole tree_grav;`
  - 色付き文字で書かれた部分が変数名
  - クラスの中にはメンバ変数、メンバ関数だけでなく「メンバ形名」も持てる、`::Monopole`がその例

STL (Standard Template Library)

- とりあえずiostream（入出力）とvector（可変長配列）ぐらいを押さえておけば大概のことができる
  - 入出力に関してはC言語のもの(cstdio)を使ってもいい（FDPSを使う分にはどちらでも可能）

```
#include <vector>
int main(){
    std::vector<int> array;
    array.resize(10);
    for(int i=0; i<10; i++){
        array[i] = i;
    }
    return 0;
}
```

```
std::vector<FP> ptcl;
for(...){
    FP ith;
    ...;
    ptcl.push_back(ith);
}
```

サンプルコードの初期条件生成（簡略化）

まとめ

- FDPSを利用するにあたって抑えておきたいC++の機能
  - 名前空間
    - `PS::`と`std::`
  - メンバ関数を持ったクラス（構造体）
    - 粒子クラス（構造体）はユーザー定義
    - FDPS提供クラスのメンバ関数をユーザーが呼び出す
  - テンプレート
    - 用意されたものを使うことができれば十分
    - FDPS提供のクラステンプレートに<>でユーザー定義の粒子クラス（構造体）を渡して実体化
  - 標準ライブラリ
    - IOはCのものかC++のもの（あるいは混在）
    - std::vectorを可変長配列として使えるぐらいで十分

C++をFortranと比べて

- 性能：どちらも速度重視の言語なので最近は大きな差がつくことは少ないと思います
- ユーザー数：C++の方がずっと多いけど数値計算をする人はそのごく一部、Fortranユーザーは少ないけれどその殆どが数値計算の人です
- 教科書：
  - Fortranは和書が限られる、2003/2008だと洋書に。ただし教科書は数値計算のためのもの
  - C++の教科書は数は多いけれど数値計算をしたい人の為に書かれているわけではない。
    - 「オブジェクト指向言語C++を学ぶにあたってはオブジェクト指向を理解してオブジェクト指向で設計と開発ができるよう・・・」
    - 数値計算がしたいのであってオブジェクト指向に入信したいわけではない、というかオブジェクト指向が数値計算に役立つ気がしない