FDPSを使うためのC++

FDPS講習会2019

野村昴太郎

はじめに

- 世界の計算機センターでサポートするプログラミング言語
  - 事実上C/C++とFortranの二択
- FDPSは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」という機能を多用しているため
- 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）
  - Cインターフェースを通じていろいろな言語から使う(今回はなし)
- ここでは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
- FDPSのサンプルコードを困らず読めるくらいを目標に

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」の利用等を優先してC++11までを使用
  - C++11以降に対応するコンパイラを使えるのであれば，ユーザーコードでは最新の仕様も使える
  - （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る）

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＋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
  - std::vectorやstd::iostreamなど
  - Cのstdioなどでもよいので使えなくても大丈夫

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み
  - 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の名前空間定義は特に必要無い
- サンプルコードにでてくる`PS::`や`std::`がわかればよい

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++クラス
  - ```
    class Vector3{
      public:
        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
    - デフォルトがprivateかpublicかだけ違う

クラス（構造体）の使用例

```
class F64vec{
  public:
    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;
    }
};

class Particle{
  public:
    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「メンバ型名」も持てる、`::Monopole`がその例

テンプレートの例

```
template <class T>
class A{
 public:
 T var;
};

template <class T>
T add(T a, T b){
  return a+b;
}

int main(){
  A<int> hoge;
  A<double> huga;
  hoge.var = add<int>(1,2);
  huga.var = add(1.0,2.0);
}
```

```
class A_int{
 public:
 int var;
};
class A_double{
 public:
 double var;
};
int add(int a,int b){return a+b;}
double add(double a, double b){
  return a+b;
}

int main(){
  A_int hoge;
  A_double huga;
  hoge.var = add(1,2);
  huga.var = add(1.0,2.0);
}
```

左と右は同等のコード．この例では型ごとにAやaddを作らなくても良くなる．
FDPSではユーザーが定義する粒子データ型に対して様々な操作を行うために利用

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 <iostream>
int main(){
  std::cout << ”Hello, world!” << std::endl;
}
```

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

配列を用意して値を代入

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としてstd::iostream，可変長配列としてstd::vectorを紹介