C++について

細野 七月

京都大学大学院総合生存館特任助教/理化学研究所計算科学研究機構客員研究員

2017/03/01 PCoMS東北大学 HPC技術講習会 FDPS 講習会

背景(なんでこんな話をするか)

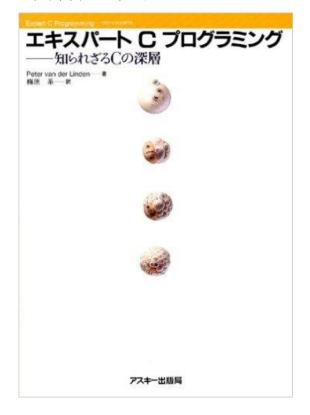
- 最近の計算機センター事情
 - 言語としてはFortran/C++の実質2択
 - プラス並列化としてMPIとOpenMP
 - 数値計算業界ではやはりFortranのシェアが大きい
- FDPS (Framework for Developing Particle Simulators)
 - C++で書かれている
 - ユーザーコードもC++で書くことを当面は想定
 - ユーザーにC++の高度な知識は要求しない
 - MPIとOpenMPによる並列化はFDPS側で面倒を見る
 - が、C++の最低限の書き方は知っておく必要がある

習得しておきたいC++の機能

- 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)
- C++の全ての言語機能を使おうとはしないこと
 - どんな上級者でも迷宮入りします

教科書とか

- 何が「良書」かというのは非常に難しい
 - 言語機能が多すぎてコーディングスタイルが多様
- C言語を経由してC++を知りたいひとには以下が名著



chapter 11 C言語を知ってりゃ、C++も簡単!

くたばれオブジェクト指向!	_
抽象化――大事な機能だけの抽出 ――――――――――――――――――――――――――――――――――――	
SD 重要な概念: 抽象化 284	
カプセル化――関連する型とデータ、関数のグループ化 ――――――――――――――――――――――――――――――――――――	_
SD 重要な概念――クラスがコードとその関連データをカプセル化(一体化)する 286	
クラスの紹介――ユーザー定義の型に、基本データ型と同じ能力を持たせる ――――	_
SD 重要な概念: クラス 287	
アクセス指定子	_
宣言	_
PC C++のプログラムのコンパイルと実行 289	
PC メソッドの本体を書く 290	

一緒にFDPSを開発したい人向け



- ユーザーには不要です
- FDPSの実装にはとても役立 ちました
- 興味本位で閲覧してみるのも、、、

言語のバージョン等

- 多くのプログラミング言語にはバージョンがあります
 - FORTRAN/Fortranだと66, 77, 90, 95, 2003, 2008等
 - C++には98, 03, 11, 14
 - 便利な新機能が増えることもありますが、新しすぎるとコンパイラの対応が追いつかないことも
- FDPSはC++03で実装してあります
 - 「京」でC++11はまだのようなので
- ユーザーコードはC++11/14で、というのも可能です
 - 「京」での使用も考えている方は03に留めるのが無難です
- GNU、Fujitsu、Clangぐらいの環境ではサンプルコードが通ることを確認するようにしていますが、中でテンプレートを多用しているので環境やバージョンによっては問題が発生するかもしれません。
 - 環境やバージョン等でコンパイルに問題が発生したときはサポートまでご 一報お願いします

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は実質的に同じもの

クラス(構造体)の使用例

```
template <typename T>
struct Vector3{
    T x, y, z;
    const Vector3 &operator+=(const Vector3 &rhs){
        x += rhs.x; y += rhs.y; z += rhs.z;
        return (*this);
    friend Vector3 operator*(const T &s, const Vector3 &v){
        Vector3 t = \{s*v.x, s*v.y, s*v.z\};
        return t;
};
typedef Vector3<double> F64vec;
struct Particle{
    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

メンバ関数をグローバル関数 から呼び出してみるテスト

コンストラクタ/デストラクタと変数の寿命

特別なメンバ関数: コンストラクタ:変数の生成時に呼ばれる #include <iostream> デストラクタ :変数の消滅時に呼ばれる struct Life{ const char *name; Life(const char *_name) : name(_name){ std::cout << "begin " << name << std::endl; | コンストラクタで文字列を表示 ~Life(){ デストラクタで文字列を表示 std::cout << "end " << name << std::endl;</pre> **}**; static Life global("global"); 実行結果 begin global int main(){ begin immediate Life("immediate"); immediate end Life outer ("outer"); begin outer begin inner

end

end

end

inner

outer

global

Life inner("inner");

return 0;

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>::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;
}</pre>
```

参照渡し

- C言語では関数に渡した値を 書き換えて欲しいときはポインタを渡す必要があった
- C++では参照渡しという機能 が追加された、FORTRANに 近い書き方になった
- 配列もvectorの(constな)参照 で渡せばいいので、ポインタ 渡しを使う機会は少ない

```
// ポインタ版
void copy(double *dst, const double *src){
    *dst = *src:
// 参照版
void copy(double &dst, const double &src){
    dst = src;
}
int main(){
    const double src=0.0;
    double dst;
    copy(&dst, &src); // ポインタ版
    copy(dst, src); // 参照版
    return 0;
```

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
 - 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(STL)
 - 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++の全ての言語機能を使おうとはしないこと