定例報告@5/16

中村研究室 B4 河田鷹介

平成 20 年 5 月 15 日

ここ 2 週間にやったこと

- フラグメントセパレータのシミュレーション (フェルミ運動を考慮)
- フラグメントセパレータの C++への移植
- ラザフォード散乱の軌跡を描くプログラム (C++)
- 中性子検出器の検出効率 (進行中)

フラグメントセパレータの C++への移植

成功。ただし同一のパラメータで実験した場合にも、FORTRAN と C++とで結果に若干の差がある。

C++の長所と短所

長所

プログラムをオブジェクティブに書ける

変数の「型」の概念をより進めて、幾つもの異なる型の変数や関数を集めて 1 つのまとまりとして定義したものを「クラス」と呼ぶ。
”Nucleus” というクラスを定義したとする。クラスはそのままではプログラム内で何の働きもしないので、実際に使うためには「オブジェクト」にする。

Nucleus Ar40(40,18,95.0);

これで、Nucleus クラスの「インスタンス」として Ar40「オブジェクト」が生成された。これが名実共に Ar40 として振る舞うためには、そうなるように Nucleus の定義をしておかなければならないが、うまく定義しておけば、パラメータを変えるだけで様々な原子核のオブジェクトを得ることができる。
なお、上のコード例で Ar40 の後にカッコがついているのは、「コンストラクタ」と呼ばれる特殊な関数である。これは、クラスからオブジェクトが生成される時に呼び出され、オブジェクトに個性を与える処理を行う。

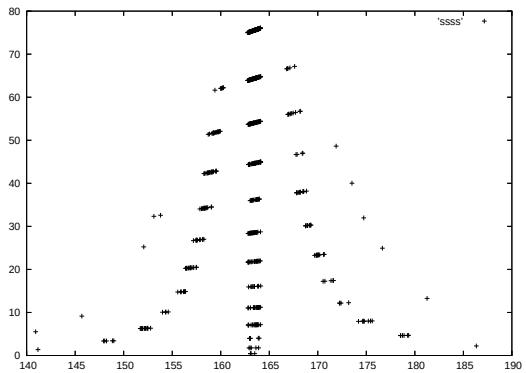

図 1: ToF:ELoss

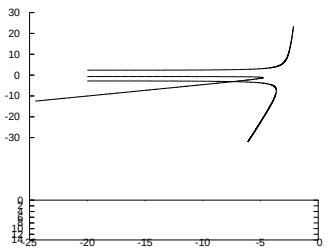

図 2: He の Au によるクーロン散乱

演算子のオーバーロード

上では物理的主体をオブジェクトとして扱う例を示したが、数学的な構造をオブジェクト化することもできる。複数の変数を 1 つにまとめて扱う事を C 言語や Fortran90 以降などでは構造体と呼ぶが、例えば double(real*8 のこと) 変数を 3 つ集めて 3 次元ベクトルの構造体を作っても、その" ベクトル" がどう振る舞うかは構造体の外で定義しなければならない。
また C++においても、通常は + や ∗ などの演算子はスカラーに対しての演算しかしない。
しかし、適切な (関数の定義と殆ど同じ) 手続きをすることで、既存の演算子を特定のオブジェクトに対して働かせるようにすることができる。

C++を使用したツールが増えてきている これが最も大きなメリットかもしれない。例えば Geant4 など。

途中で物理的主体が増えるような過程をシミュレーションするときには、新たなオブジェクトを動的に作れる機能が役に立つはずである。また、クラスや演算子の定義を適切に行えば、自然な記述で物理現象をプログラムコードに落とし込むことができるようになり、これはバグや論理的ミスの予防に役立つ。

短所

遅い FORTRAN の実行速度に比べるとどうしても劣る。

オブジェクティブにやるのが面倒 オブジェクトの雛型となるクラスの定義などをしっかりしなければならない。ファイルを分割して整理しようとすると更に面倒。

C++から FORTRAN を使うのが面倒 FORTRAN に変数を渡すときは参照渡しでなければならず (= リテラルは渡せない)、更に前もってキャストしておかなければならない。FORTRAN で書かれた (特に古い) ルーチンを利用する際はこの点がかなりの障害になる。
また、C++ソースと FORTRAN ソースは別々にコンパイルした上で*FORTRAN のリンカで*リンクしなければならないことに注意。

以上の理由から、ある程度大規模なシミュレーションでないと、積極的に C++を使う理由は見出せない。

実例

Mag = *Exp1*.TuneMagnet(*Mag*, *Be*, *Ar40*, 1000, 24, 8);
斜体がオブジェクト。Mag は磁石、Exp1 は実験 (という抽象的なものもオブジェクトになりうる)、Be は壁物質,Ar40 は原子核。Exp1 の関数 TuneMagnet は、磁石オブジェクト、壁、壁に入射する原子核の種類と数、選別したい原子核の A と Z を入力すると、入力条件下で目的の原子核を選別するのに最適な磁場を持つ磁石オブジェクトを返す関数。適当に定義した磁石をこの関数で実験に適するように調整する。

```
inline double operator * (const mathvector3 &v1, const mathvector3 &v2){
  double dotprod=0;
  for(int i=0; i<3; i++) dotprod += v1.element[i]*v2.element[i];
  return dotprod;
}
```

斜体がオブジェクト。v1,v2 は mathvector3 オブジェクト。これは数学的な 3 次元ベクトルとして定義されたもので、このコードは、mathvector3 オブジェクト同士の乗算、つまり内積の演算を、一般的な乗算演算子 ∗ に持たせる手続きを行っている。このコードにより、C++コンパイラは ∗ の両側が mathvector3 オブジェクトである場合、通常の乗算の代わりに上で定義した処理を行うよう ∗ の働きを変える。

今後

- 中性子検出器の検出効率を求めるプログラム (neut) の理解
- Geant4 を実際に少し使ったプログラムの作成