C++チュートリアル

2014年度

河野能知

目的

- 素粒子物理分野で良く使われている解析ツール（ROOT）を使えるようになること

- ROOT
  - http://root.cern.ch
  - ヒストグラム、大量データ解析のための枠組み、統計分析、…
  - ROOT自身C++で書かれていて、様々なツールは専用のクラスとして実装されている
  - C++コードを記述してこれらのクラスを使用する
  - 用意されていない機能は自分で開発したコードを組み込むことで対応

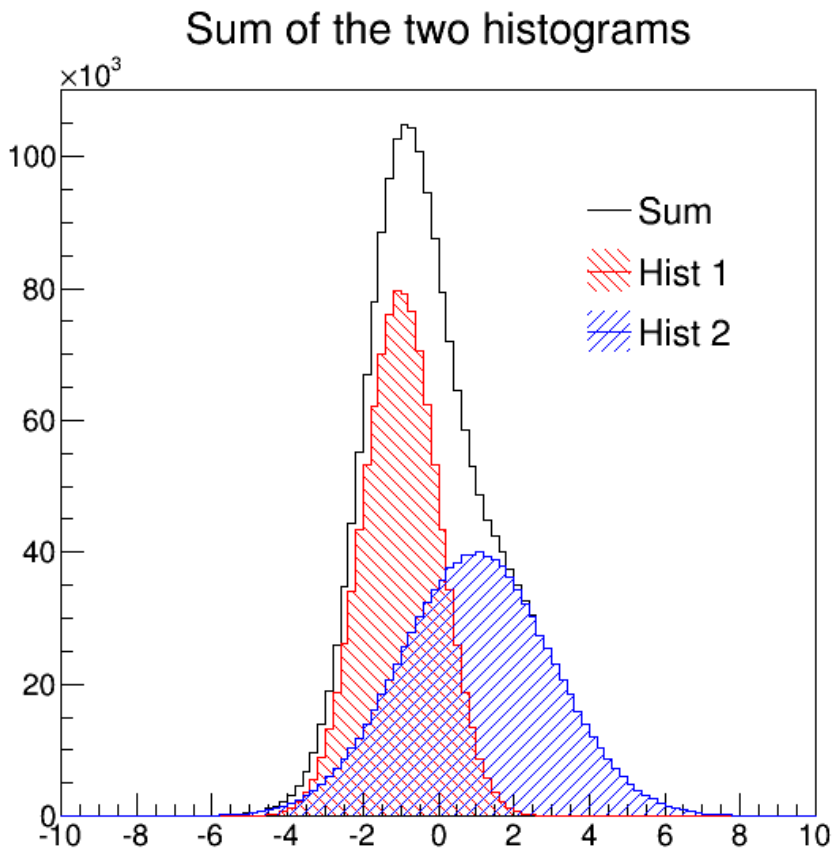
- 必要に応じて自分でC++コードを書けるようになる
  - まずは100行くらいのコードを掛けるようになることを目標に。

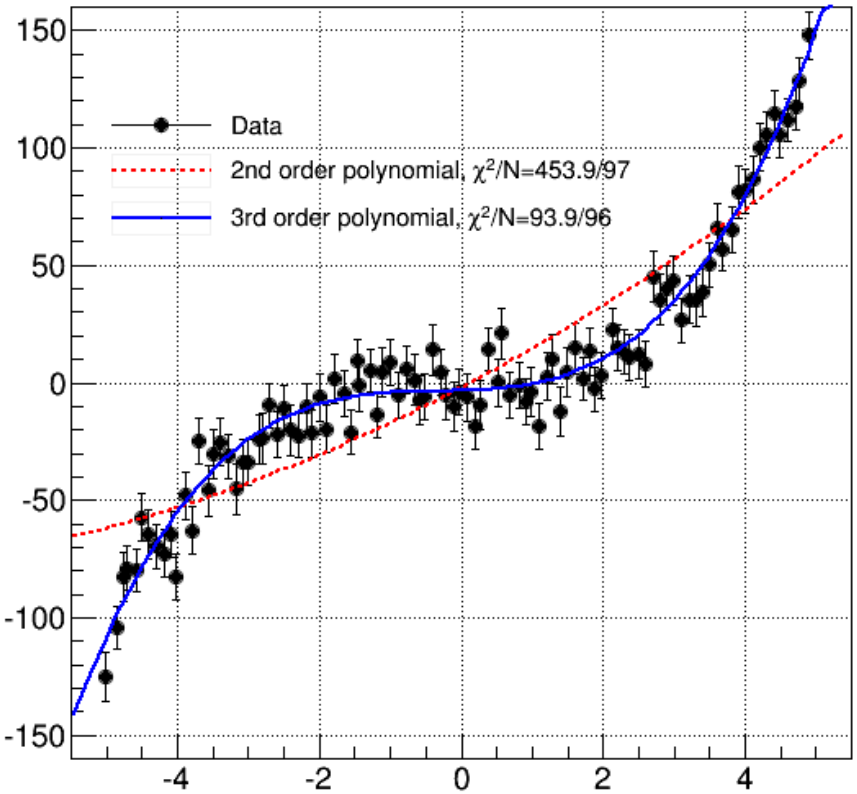
内容

- コンピュータ・プログラミングの開発と実行
  - Linux上での作業に慣れる
- プログラム例を通していろいろ学ぶ
  - 実際にコードを書いて走らせて慣れることが重要
  - 例として物理の問題を解いてみる（古典的な粒子の運動）
- 文法事項
  - ソースコード、コンパイル、実行可能ファイル、ライブラリ、Make
  - データ型、演算子、入出力
  - 条件分岐、繰り返し
  - 関数、配列、ポインタ、参照
  - データ構造、クラス
  - テンプレート
  - このような要素があるということで、具体的にはコード例を参照

ROOTで良く行うこと(1)

複数の分布を比較する





分布を関数でフィットする

他にもいろいろ、http://root.cern.ch/drupal/category/image-galleries/data-analysis-visualization

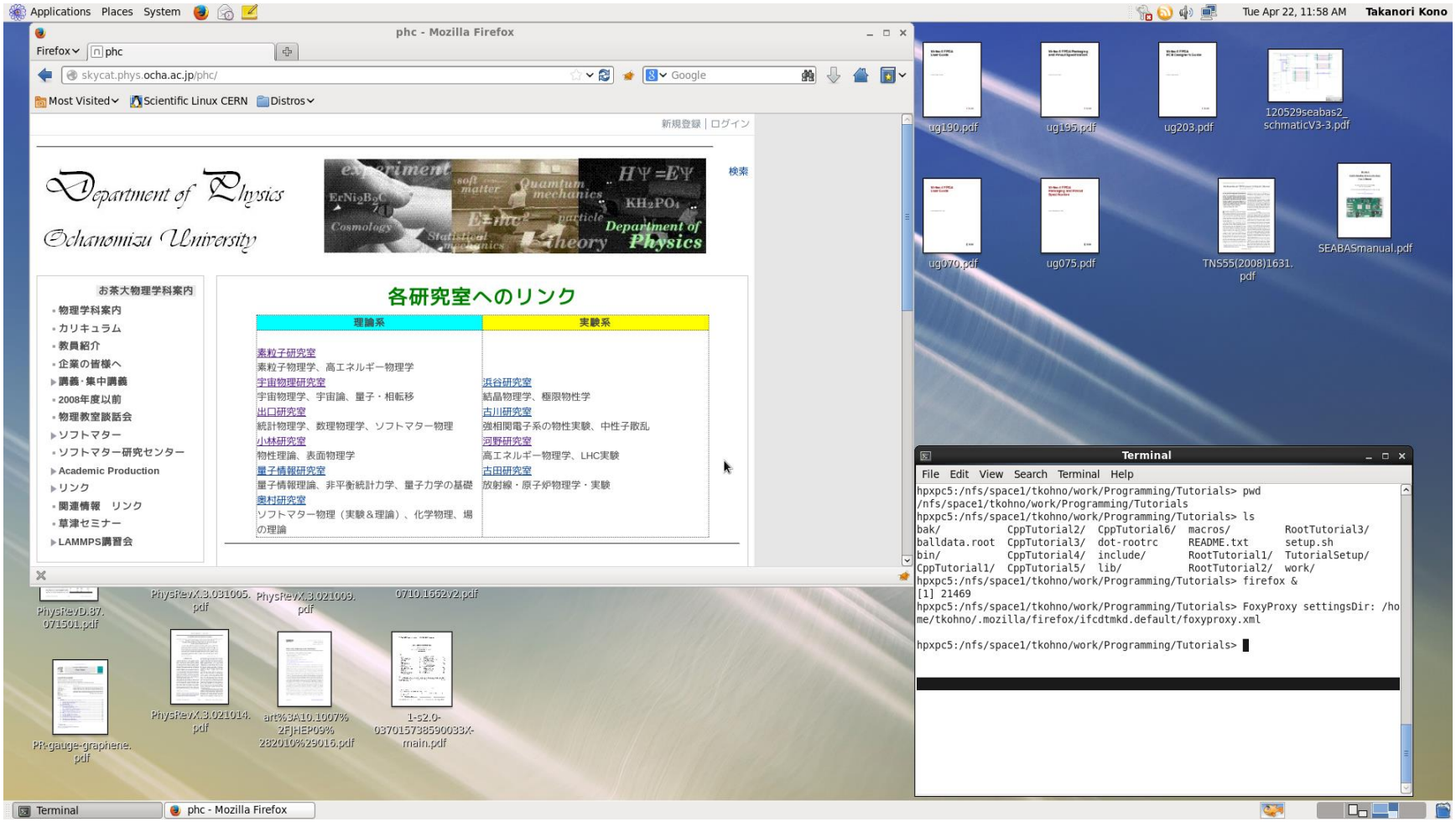
Linux環境

Linux環境におけるプログラミング

- プログラミングはLinux上で行うことを前提とする
  - 無料のコンパイラ（GNUコンパイラ）
  - 様々なツールを自由に組み合わせて使える

ツール名	コマンド名	用途
Emacs	Emacs	テキストエディタ。ソースコードの編集
g++	g++	C++コンパイラ（Cの場合はgcc）
Make	make	コンパイル作業の効率化のために使用
Subversion	svn	ソースコードのバージョン管理。サーバーを用いて複数の人でコードを共有するために使用
Terminal	gnome-terminal	コマンドを入力する画面

Linux環境


Applications Places System
Tue Apr 22, 11:58 AM Takanori Kono
phc - Mozilla Firefox
skycat.phys.ocha.ac.jp/phc/
Department of Physics
Ochanomizu University
各研究室へのリンク
理論系
実験系
Terminal
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> pwd
/nfs/space1/tkohno/work/Programming/Tutorials
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> ls
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> firefox &
[1] 21469
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> FoxyProxy settingsDir: /home/tkohno/.mozilla/firefox/ifcdtmkd.default/foxyproxy.xml
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials>

Linux PCへログイン

- Linux PC名
  - hpxr1.phys.ocha.ac.jp
- 他の計算機から、このPCに接続するにはsshというプログラムを使う
- Macのターミナル上で次のコマンドを実行
  - ssh –XY <ユーザー名>@hpxr1.phys.ocha.ac.jp

Linuxの使用方法

- WindowsやMacは、マウスを用いて視覚的に操作できるものが一般的
- Linuxにおける作業の基本は、コマンド入力によるもの
  - ターミナル上で、コマンドを実行する

Terminal

File Edit View Search Terminal Help

```
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> pwd
/nfs/space1/tkohno/work/Programming/Tutorials
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> ls
bak/            CppTutorial2/  CppTutorial6/  macros/         RootTutorial3/
balldata.root   CppTutorial3/  dot-rootrc     README.txt      setup.sh
bin/            CppTutorial4/  include/       RootTutorial1/  TutorialSetup/
CppTutorial1/   CppTutorial5/  lib/           RootTutorial2/  work/
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> firefox &
[1] 21469
hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials> FoxyProxy settingsDir: /ho
me/tkohno/.mozilla/firefox/ifcdtmkd.default/foxyproxy.xml

hpxpc5:/nfs/space1/tkohno/work/Programming/Tutorials>
```

pwd: 現在の作業ディレクトリを表示

ls: 現在のディレクトリ上のファイルを表示

firefoxを起動

裏でシェルと呼ばれるプログラムが動作していて、入力したコマンドに応じて適切なプログラムを呼び出している

Linuxで良く使うコマンドは、「linux コマンド」で検索

コード例のダウンロードと環境設定

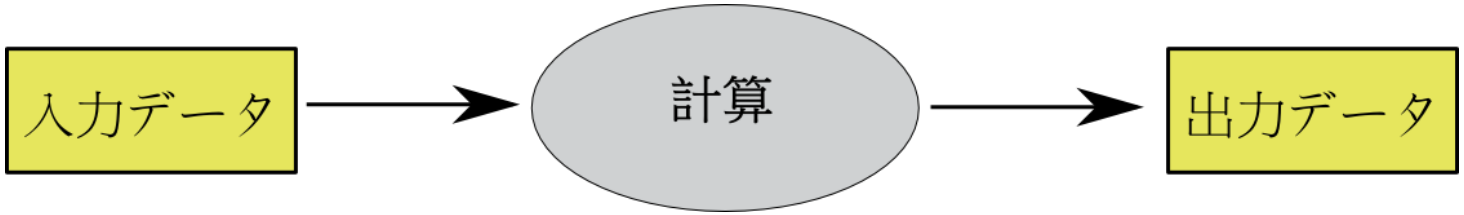
- コード例はSubversionレポジトリに保管してある
- 今のところhpx.phys.ocha.ac.jpをサーバとして使っている
- 方法
  - 作業ディレクトリを決める。（$HOME/CppTutorial等）
  - cd $HOME/CppTutorial
  - export OCHA_SVN=svn+ssh://hpx.phys.ocha.ac.jp/var/svn/repos
  - svn co $OCHA_SVN/Lab/Tutorials
  - （ファイルがたくさんダウンロードされたはず、ls –lで確認）
  - TutorialSetup/setup.shの中のCPP_TUTORIAL_DIR, CPP_TUTORIAL_WDIRを自分の作業ディレクトリを指すように書き換える
  - cd TutorialSetup; source setup.sh
- これで、各演習をコンパイル、実行できるはず
- http://hpx.phys.ocha.ac.jp/websvnからもコードを見られる

コード例の内容

ディレクトリ名	プログラム例	学習内容
CppTutorial1	HelloWorld、足し算	コンパイル、Makefile 数値型、変数、コマンドライン引数
CppTutorial2	素因数分解	繰り返し、条件分岐
CppTutorial3	微分方程式の解法 ボールの軌跡の追跡	数値積分（RungeKutta法） 分割コンパイル、共有ライブラリ
CppTutorial4	多粒子系に拡張	配列 データ構造、クラス
CppTutorial5		メモリ、ポインタ、参照
CppTutorial6		テンプレート、STL
RootTutorial1		ヒストグラム、フィット
RootTutorial2		関数、フィット
RootTutorial3	多粒子系の衝突	TTreeを用いたデータの保存、解析
RootTutorial4	素粒子反応の解析	TTree::MakeClass

簡単なプログラムの作成と実行

コンピュータ・プログラム



- あるデータを別のデータに変換（計算）
  - 1 + 2 → 3
  - 実験データ → 画面上にグラフを表示
  - URL → ホームページのデータ
- 入力データまたは出力データは無くても構わない

- 基本的なデータ、計算
  - 数値（整数、浮動小数点数）とその四則演算
  - 文字、文字列、（ビット列）
  - 繰り返し、条件分岐
- これを多数組み合わせると複雑なことができるようになる。

プログラミングの要素

- データ型
  - 整数型、浮動小数点型、論理値（true, false）、配列
- 演算子
  - 数値演算（+, -, *, /, %, <, >, <=, >=）, ビット演算（&, |, >>, <<, ~）
  - 論理演算（&&, ||, !, ==, !=)
  - 省略形（++, --, +=, -=, *=, /=, &=, !=, …）
- 論理の制御
  - 繰り返し（for, while, do-while）
  - 条件分岐（if-else）
- 入出力
  - コマンドライン引数、標準入出力（ターミナル）、ファイル、通信、描画、…
- 関数
- 構造体、クラス（複合的な型）
  - あると便利な機能を自分で追加できる

Tutorial 1： プログラムの基本

HelloWorld.cxx
- 簡単なプログラムでコンパイルして実行するまでの手順を確認
- main関数

プログラムのコンパイル

（ソースコード）

```
int main(…) {
   …..
}
```

.cxx
.cpp
.cc
.C, …

g++ <オプション> –o <実行可能ファイル> <ソースファイル>

より複雑なプログラムでソースコードを分割する方法はもう少し後で

実行可能ファイル（機械語）

.exe等

実際には何でもよい
拡張子なしの場合も多い

基本的な構文

- 変数の宣言
  - [型]　[変数名]; [型]　[変数名]=[初期値];
  - int a; int b=51; float c=3.23;
- 代入
  - a=1; a=b;
  - kansu_no_atai = some_function(1, b); // 関数に引数を渡して値を代入
- それぞれの文は「;」で終える
- 文の集まり
  - 繰り返しの内容、条件下の処理、関数の中身等
  - 一まとまりのブロックを{　}で囲む。(複数の文の集まり)
- コメントは、/* … */で囲むか、// 以降行の終わりまで
- #include [ファイル名]:　他のファイルのコードを埋め込む

HelloWorld

```
/*
  HelloWorld.cxx
  ==============
  The first example.
*/
#include <iostream>

int main(int argc, char* argv[]) {
  std::cout << "Hello, World!" << std::endl;
  return 0;
}
```

コメント

入出力に必要なコード（cout）を使用

main関数（プログラムの本体）

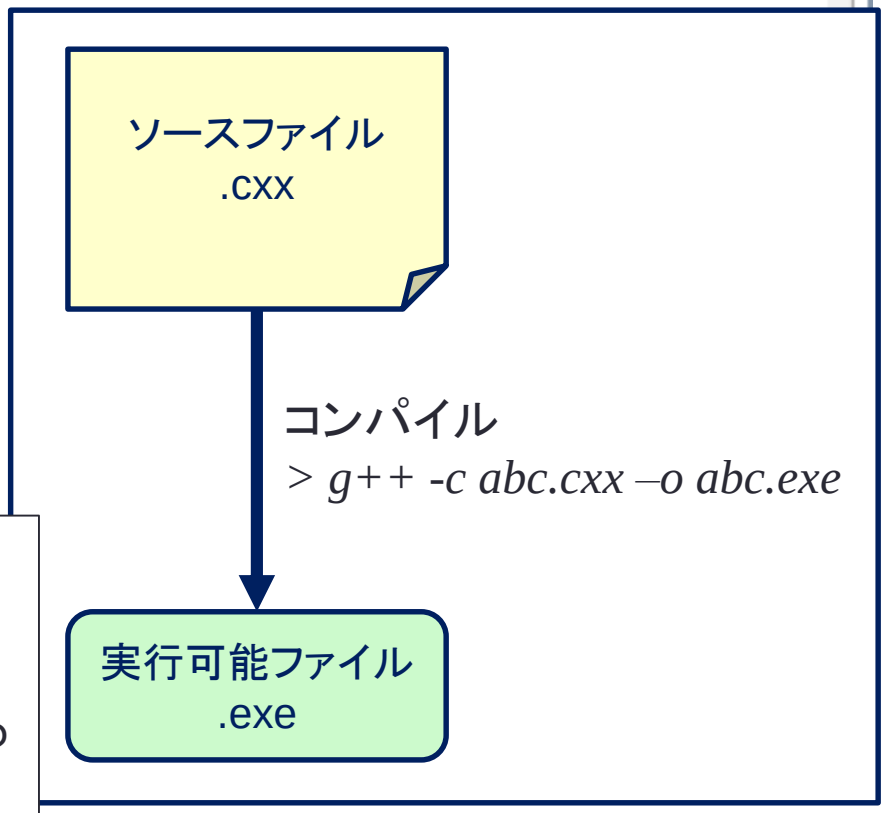
main関数の中身

makeして実行（./HelloWorld.exeと入力）すると、
Hello, World!
という文字列を画面に出力する

Tutorial1: コンパイルとMake

Makefile

```
#-----------------------------------------------------------------------
# Makefile for standalone programs
#-----------------------------------------------------------------------

PROG_SRCS = Add.cxx Echo.cxx HelloWorld.cxx Add_1plus2.cxx Add_AplusB.cxx \
        Calculate_AandB.cxx
PROG_EXES = $(patsubst %.cxx, %.exe, $(PROG_SRCS))

.PHONY: all clean

all: $(PROG_EXES)

clean:
        @rm -f *.exe *~

$(PROG_EXES): %.exe: %.cxx
        g++ -o $@ $<
```



- コンパイルのルールを記述
- <ターゲット>: <依存するファイル>
  - コンパイルの方法（コマンド）
- makeと打つだけで、これを実行してくれる
- make allで全てのファイルをコンパイル

1＋2＝3の計算と出力

```
#include <iostream>
#include <cstdio>
#include <cstdlib>

int main(int argc, char* argv[]) {
  int a=1, b=2;
  int c = a + b;
  std::printf("%d + %d => %d\n", a, b, c);

  return 0;
}
```

printf関数を宣言しているコードを埋め込む

整数型変数a,bの定義

a+bの結果を代入

書式（フォーマット）

標準出力（画面）に文字列を出力するための関数を呼び出す

“%d”の部分は、後ろに追加した変数の値で置き換わる

Tutorial 1： プログラムの書き方

Add*.cxx
- 簡単なプログラムでコンパイルして実行するまでの手順を確認
- main関数

「１＋２」という計算を行いたい時、どのようなプログラムを作るか

- １＋２のみを計算する。それ以外の計算はできない → Add_1plus2.exe
- 任意の2つの数の和を計算する → Add_AplusB.exe
- 任意の2つの数の四則演算をする → Calculate_AandB.exe

Tutorial 2： 繰り返しと条件分岐

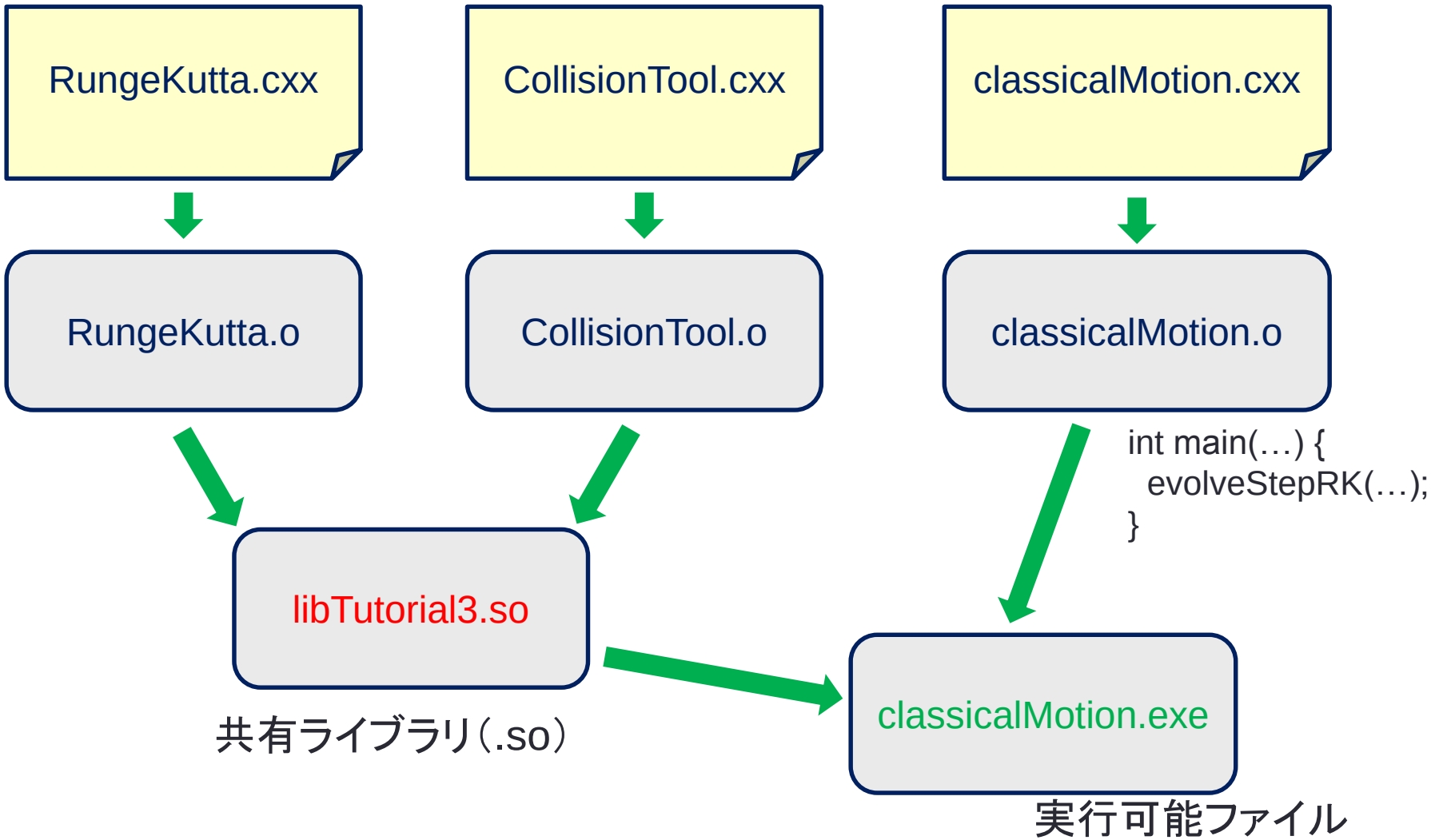
素数の判定

1. ある整数Nが素数かどうか判定する
   - 2～$\sqrt{N}$の全ての整数mに対して、N/mが割り切れるかどうかチェック
   - 一つでも割り切れるものがあれば、Nは素数ではない
   - PrimeNumber1.exe

2. ある整数N以下の全ての素数を求める
   - 1～ Nの全ての整数に対して、それが素数であるかどうかを1.の手順に従って求める
   - 最大$\mathrm{N} \times \sqrt{\mathrm{N}}$の計算量
   - PrimeNumber2.exe

力技で効率の良いアルゴリズムでは無いけど、最近の計算機では問題2も、$N \sim 10^6$くらいなら一瞬で終わる

Tutorial 3： 複数ファイルのコンパイル(1)



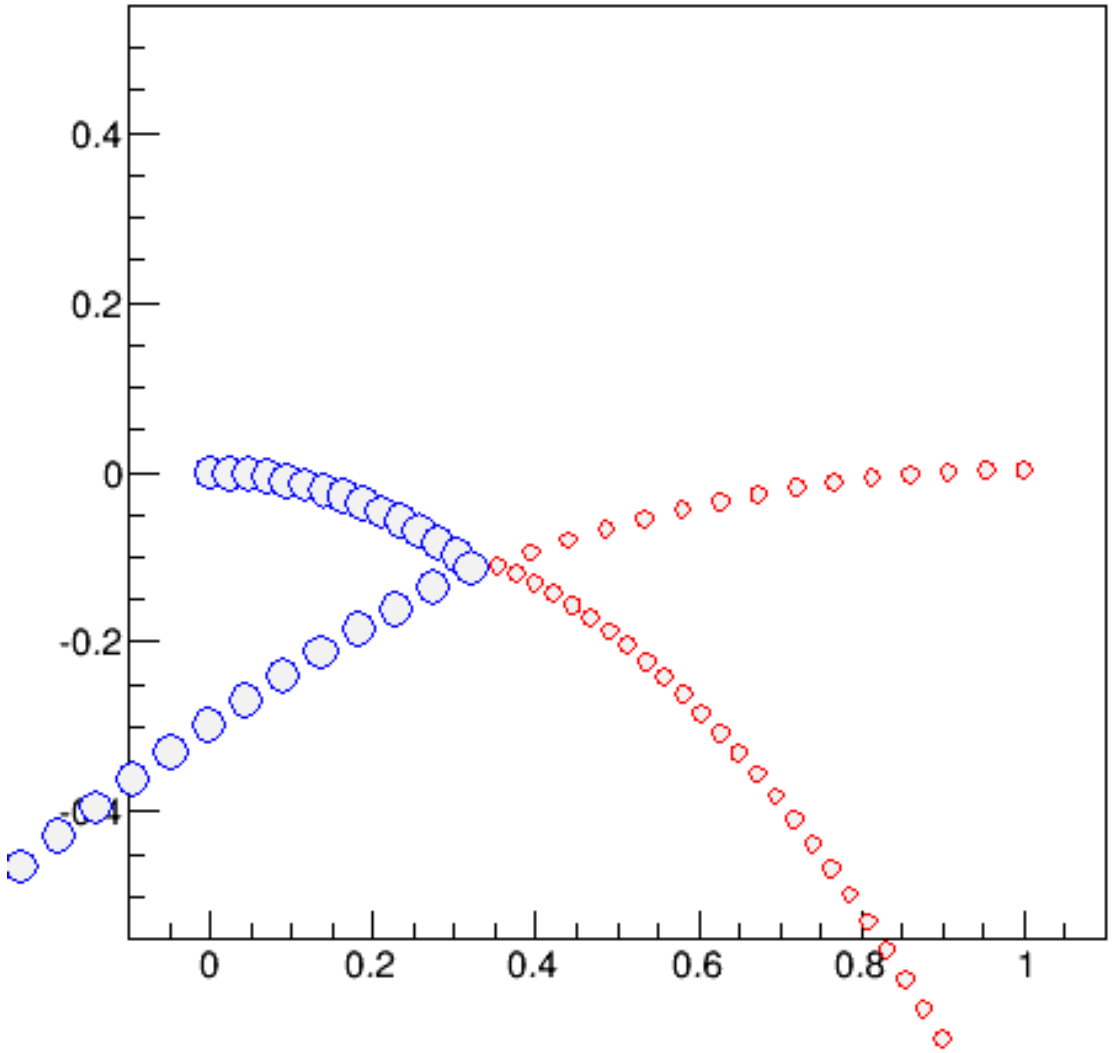
Tutorial 3： 複数ファイルのコンパイル(2)

```
PROG_EXES = classicalMotion.exe collisionOfBalls.exe

classicalMotion.o: classicalMotion.cxx
        g++ -c -I. -o $@ $<

collisionOfBalls.o: collisionOfBalls.cxx
        g++ -c -I. -o $@ $<

RungeKutta.o: RungeKutta.cxx RungeKutta.hxx
        g++ -fPIC -c -I. -o $@ $<

CollisionTool.o: CollisionTool.cxx CollisionTool.hxx
        g++ -fPIC -c -I. -o $@ $<

libTutorial3.so: RungeKutta.o CollisionTool.o
        g++ -shared -o $@ $^

classicalMotion.exe: classicalMotion.o libTutorial3.so
        g++ -o $@ $< -L. -lTutorial3

collisionOfBalls.exe: collisionOfBalls.o libTutorial3.so
        g++ -o $@ $< -L. -lTutorial3
```

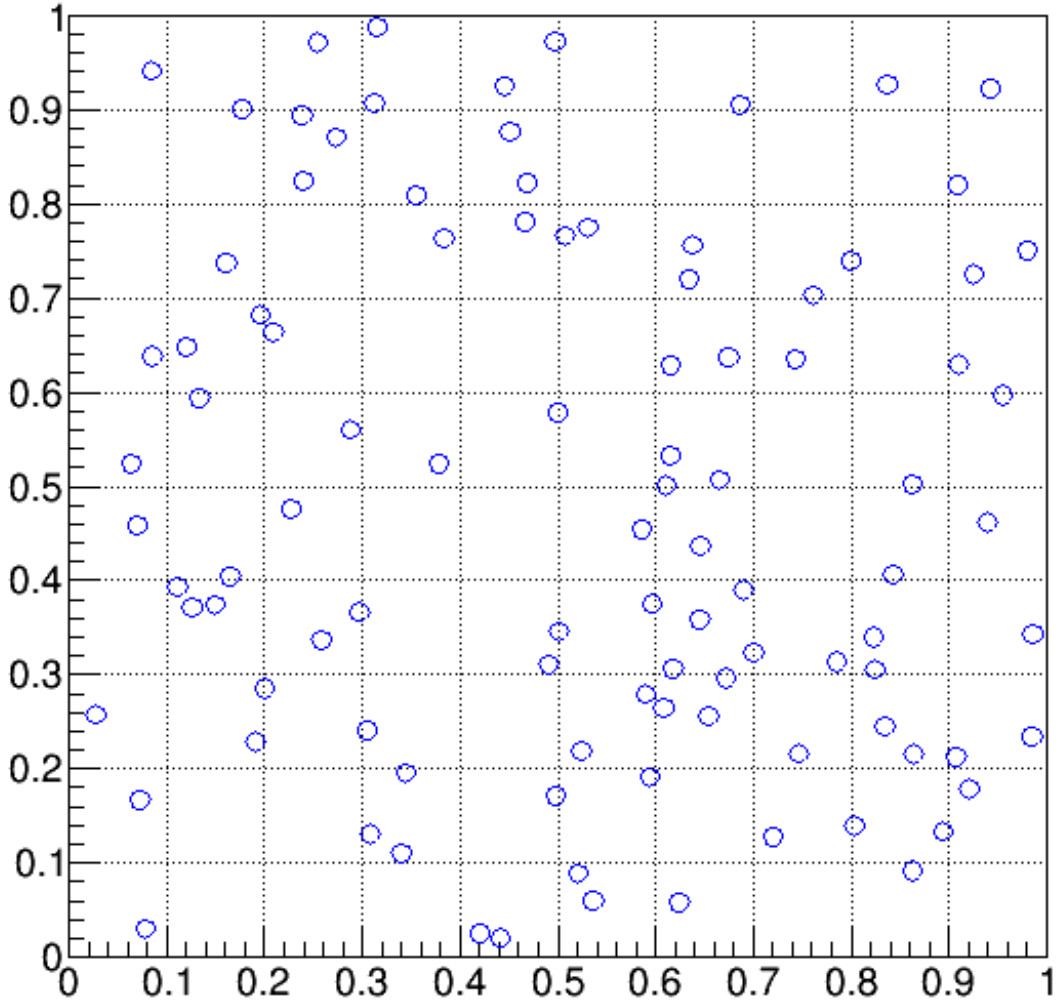
Tutorial 3: 微分方程式を解く

- 運動方程式$\vec{F} = m\vec{a}, \vec{a} = \dot{\vec{v}}, \vec{v} = \dot{\vec{x}}$を解く
- 数学的には積分すれば良い
  - $\vec{v} = \int \frac{\vec{F}}{m} dt + C_1, \vec{x} = \int \vec{v} dt + C_1$
- 計算機で扱えるように足し算、掛け算、繰り返しに置き換える
  - 積分を和で置き換える（$\int_{t_1}^{t_2} f(t)dt \to \sum_{i=1}^{N} f(x_i)\Delta t,\ \Delta t = \frac{t_2 - t_1}{N}$）
- ただし、このままでは精度があまり良くない
  - 被積分関数が時間に依存する時、近似として粗過ぎる
- 4次のRunge-Kutta法を使ってみる
  - 被積分関数のTaylor展開の4次まで考慮したもの
  - Numerical recipe参照（Chapter16）
  - 時刻t_iからt_{i+1}まで積分するのにt_iにおける関数値だけでなく、その間に含まる4点における関数値も使って精度を高める

Tutorial 3: 古典的な粒子の衝突



C++ Tutorial 4: 多数の粒子の運動



C++ Tutorial 4： 配列

- 100個の粒子の運動、衝突を追跡する
- 粒子の座標(x, y)、速度(vx, vy)を百個分定義する

```
int main(int argc, char* argv[]) {
  // Variables for the balls
  const int N=100;
  float m[N];
  float x[N];
  float y[N];
  float vx[N];
  float vy[N];
  float r=0.01; // [m]

  // Simulation parameters
  float dt=0.0002, tmax=5.0; // [s]
  float t=0.0;

  int i, j;
  bool overlap=false;
```

粒子の変数をそれぞれN個確保

C++ Tutorial 4: データ構造

- x[100], y[100], vx[100], vy[100], …

の代わりに

- Ball=(x, y, vx, vy, …)という型があれば、Ball[100]と書けて便利
- そうすれば、関数などもcheckCollision(ball1, ball2)となって扱いやすい

複合的な型（構造体（C）、クラス（C++））

- いくつかのデータをまとめた新たな型を定義

```
// Check collisions among balls after the time step. (check all combinations)
for (i=0; i<n; ++i) {
  for (j=i+1; j<n; ++j) {
    status = checkCollision(x[i], y[i], vx[i], vy[i], r,
                            x[j], y[j], vx[j], vy[j], r);
    // Need improvement. Only taking into account the first occurence
    // of the collision for each ball.
    if (status) {
      status = processCollision(x[i], y[i], vx[i], vy[i], m[i], r,
                                x[j], y[j], vx[j], vy[j], m[j], r);
      continue;
    }
  }
}
```

i番目とj番目の粒子のデータ

関数に渡す引数が増えると、読みづらいし、ミスをしやすくなる

Ballクラス

```
class Ball {
public:
  Ball();
  Ball(float m, float r);
  ~Ball();

  void setPosition(float x, float y);
  void setVelocity(float vx, float vy);

  float mass() const { return mMass; }
  float radius() const { return mRadius; }
  float x() const { return mX; }
  float y() const { return mY; }
  float vx() const { return mVx; }
  float vy() const { return mVy; }

  bool overlap(const Ball& ball2);

  bool checkCollision(const Ball& ball2);

  bool processCollision(Ball& ball2);

  void print(int i);

private:
  float mMass;
  float mRadius;
  float mX;
  float mY;
  float mVx;
  float mVy;
};
```

“Ball”という名前のクラス（型）を宣言

Ballを扱う時に良く使う処理を関数として用意しておく（*メンバー関数*）
（メソッド（method）と呼んだりもする）

予め用意した関数のみを使うため、想定外の使い方が減りエラーが少なくなる

Ballクラスを使う際は詳細は見えない

Ballの属性（*メンバーデータ*）

C++ Tutorial 5: メモリとポインタ

- Tutorial4で扱った粒子100個からなる系では、粒子数はあらかじめ決まっていて、プログラムの中で100個分の配列を用意していた

- **問題**
  - 粒子数を実行時に決められるようにできないか
  - より汎用的なプログラムになる
  - 但し、プログラムを組む段階で配列の大きさが分からない
    - 実行時に必要に応じて配列の大きさを変更できれば良い

- 実行時に新たにメモリ領域を確保する
  - 演算子newを使う
  - 取得したメモリを解放するには、演算子deleteを使う

C++ Tutorial 6： テンプレート

C++ Tutorial 6： STL