SATOROOT

松本 悟

February 8, 2014

0.0 前書き

0.0.1 どんな人向け

- 実験系の研究室に配属されてデータの解析をする段階になった人
- 先輩に「ROOT 使えるようになっといてね」とか言われちゃった人

そんな人の為の覚え書き。SATOROOT とは、本ドキュメントの前身を私の後輩が作業する為に作成していたディレクトリ名から拝借した名前である。

0.0.2 方針

とりあえず動かす。C++の細かいお作法とか正しい言葉の使い方とかは無視。動かす上で必要なお作法やおまじないについてはその都度紹介したりしなかったりする。とにかく動かせるようにすることを目指す。ただし、自分で調べることにも重きを置くのでサンプルを示したらその都度サンプルをいじる練習問題を提供する。

0.0.3 作業環境

著者の作業環境は

- OS X 10.8.5
- ROOT version 5.34/09

0.0.4 お約束事

- $ — プロンプトを表す記号。パソコンがユーザーの入力を受け入れる状態を表す。
- root[i] — i には数字が入る。コマンドライン上で ROOT を作業している時の行番号である。i を省略することもある。
- SATOROOT — この覚え書きで使用する全てのファイルは SATOROOT 以下のディレクトリで行う。

Contents

0.0 前書き i
0.0.1 どんな人向け i
0.0.2 方針 i
0.0.3 作業環境 i
0.0.4 お約束事 i

1 ざっとみる C++ **2**
1.1 何もしないプログラムを作成・実行する 2
1.1.1 何もしないプログラムを用意する 2
1.1.2 初めてのコンパイル 2
1.2 Hello, world 2
1.3 変数の宣言 3
1.3.1 変数の型 3
1.4 しつこり Hello, world 3

2 マクロを使う **4**
2.1 ROOT 4
2.1.1 ROOT とは 4
2.1.2 なぜ ROOT 4
2.1.3 ROOT のインストール 4
2.2 ROOT をコマンドラインで使う 6
2.2.1 ROOT を起動する 6
2.2.2 ROOT を終了する 7
2.3 ROOT でマクロを読む/実行する方法 8
2.3.1 hello.cpp の実行方法 1 8
2.3.2 hello.cpp の実行方法 2 8
2.3.3 hello.cpp の実行方法 3(推奨) 8
2.4 幾つかの常套手段 10
2.4.1 引数を数字にする 10
2.4.2 引数を文字列にする 10
2.4.3 引数を複数にする 10
2.5 関数を描く 12
2.5.1 練習 13
2.5.2 解答例 13
2.6 ヒストグラムを描く 15
2.6.1 練習 15
2.6.2 解答例 16
2.7 TRandom と TCanvas 18
2.7.1 練習 18
2.7.2 解答例 19
2.8 TGraph 20
2.8.1 ファイルから読み込んでグラフ化する 20
2.8.2 ランダムウォーク 23

2.9 任意の関数に従うヒストグラムを描く . . . 25
2.9.1 練習 . . . 25
2.9.2 解答例 . . . 27
2.10 File への出力 . . . 29
2.10.1 練習 . . . 29
2.10.2 解答例 . . . 30
2.11 File からの入力 . . . 31
2.12 tree に出会う . . . 32
2.12.1 meettree.cpp を実行する . . . 32
2.12.2 Tree を扱う . . . 33
2.12.3 Tree からヒストグラムを描く . . . 33
2.12.4 練習 . . . 35
2.12.5 解答例 . . . 35
2.13 Tree から読んで描く . . . 35
2.13.1 練習 . . . 35
2.13.2 解答例 . . . 36

3 ネイティブプログラミング **37**
3.1 ネイティブプログラムへの移行準備 . . . 37
3.1.1 プログラミング言語とコンパイラ . . . 37
3.1.2 コンパイラ c++ . . . 37
3.1.3 コンパイルする . . . 37
3.1.4 オブジェクトファイル . . . 38
3.2 Makefile その 1 . . . 38
3.2.1 Make(メイク) . . . 38
3.2.2 Makefile . . . 39
3.2.3 直打ちメイクファイル . . . 40
3.2.4 マクロを利用したメイクファイル . . . 41
3.2.5 オブジェクトファイルを意識したメイクファイル . . . 41
3.2.6 内部マクロを用いたメイクファイル . . . 41
3.2.7 ターゲット clean . . . 42
3.3 Makefile その 2:ファイルを分割する . . . 42
3.3.1 分割したファイルのコンパイル . . . 44
3.4 Makefile その 3:引数を利用する . . . 46
3.4.1 とりあえず動かす . . . 46
3.4.2 練習 . . . 48
3.4.3 解答例 . . . 48
3.5 Makefile その 4:ROOT のネイティブプログラミング . . . 49
3.5.1 ROOT ライブラリとのリンク . . . 49
3.5.2 幾つかのサンプルプログラム . . . 51

A 補足 **53**
A.1 名前空間 . . . 53
A.1.1 名前空間 std:: . . . 53
A.1.2 名前空間 TMath:: . . . 53
A.2 ROOT で使う色 . . . 54
A.3 ROOT で使うスタイル . . . 54

Chapter 1

ざっとみるC++

以下は作業中。
　C 言語もしくはその流れを汲む C++を初めて触れる人の為の最低限を紹介する。人間が書いたソースコードを機械が理解出来る言語に変換することをコンパイルという。C++のソースコードをコンパイルする方法を紹介する。

1.1　何もしないプログラムを作成・実行する

1.1.1　何もしないプログラムを用意する

まずは何もしない空っぽのプログラムを書いてみる。

empty.cpp

```
int main(){
}
```

empty.cpp は C++で書かれた本当に何もしないプログラムである。

1.1.2　初めてのコンパイル

まずは自分の作業環境にコンパイラが存在することを確認する。本テキストでは一貫して c++を用いる。empty.cpp をコンパイルする方法は以下のようである。

```
$ c++ empty.cpp
```

すると、a.out というファイルがコンパイルしたディレクトリに作成される。ただし、empty.cpp の内容を正しく入力出来ていない場合にはエラー分が出てくるのでその指示に従ってなおす。a.out の実行方法は

```
$ ./a.out
```

とすればよい。先に述べた通り、何も起きない。

1.2　Hello, world

プログラムに関わる人にはおなじみの Hello, world を出力するプログラムを作成する。

hello.cpp

```
#include <iostream>
int main(){
  std::cout << "Hello, World" << std::endl ;
  retrun 0 ;
}
```

各行の簡単な説明を行う。

```
#include <iostream>
```

C 言語はライブラリと呼ばれるものを読み込むことで、そのライブラリ内で定義されている様々な操作を行うことが可能となる。この' 読み込む' ということが #include であり、' 読み込まれるライブラリ' が iostream である。

```
int main(){
```

C 言語で書かれたプログラムが実行されたとき、一番最初に実行する関数の名前は main という関数である。この一行はその定義文である。

```
std::cout << "Hello, world" << std::endl ;
```

行末のセミコロンは C 言語では改行を意味している。std::cout は続く値や文章を標準出力に出力することをあらわす。なお std::は名前空間という概念 (A.1) が関係するが後々勉強すればよい。感覚的には std::cout という標準出力にむけて<<という記号で持って、文字列"Hello, world"を送り出している。その後、続けて std::endl を送り出している。std::endl は改行を意味する。

```
return 0 ;
```

main 関数の実行結果を OS に伝えている一文であり、0 は正常終了を意味する。0 以外は異常終了を意味する。一般的には return 1; などして明示的に異常終了を伝える。

```
}
```

main 関数に対応する括弧である。
最後にコンパイルして、実行してみよう。

```
$ c++ empty.cpp
$ ./a.out
Hello, world
```

1.3 変数の宣言

1.3.1 変数の型

C 言語には以下の様な型が存在する。以下はそれぞれどの程度のデータを確保できるかというデータサイズが決まっているが、使用環境にもよる。ここでは紹介程度ですますので、より詳しいことが知りたい人は各自で検索。

型	何に使うか
char	文字列
int	整数 (小数は扱えません)
float	数字
double	数字 (float よりも高い精度)

Table 1.1: C 言語の型

1.4 しつこり Hello, world

Chapter 2

マクロを使う

2.1 ROOT

2.1.1 ROOT とは

ROOT(http://root.cern.ch/drupal/) とは、高エネルギー業界で広く普及している膨大なデータを効率的に扱うためのフレームワークです。C++のお作法でプログラミングします。コマントライン上で ROOT と対話的にプロットやプログラミングを行うことが出来ます。

2.1.2 なぜ ROOT

世の中のいろんなニーズに応えた結果です。(投げやり)

2.1.3 ROOT のインストール

ROOT のインストール作業を行う。

- /usr/local/hep/root/5.34.09 — ROOT のライブラリ置き場
- /tmp — コンパイルを実行する時の場所

各ディレクトリの作成

```
$ sudo mkdir -p /usr/local/hep/root/v5.34.09
$ mkdir ~/tmp
```

ROOT のソースコードのダウンロードと展開

```
$ cd ~/tmp
$ sudo wget ftp://root.cern.ch/root/root_v5.34.09.source.tar.gz
$ ls
root_v5.34.09.source.tar.gz
$ sudo tar zxvf root_v5.34.09.source.tar.gz
$ ls
root
root_v5.34.09.source.tar.gz
```

環境変数の定義

```
$ export ROOTSYS=/usr/local/hep/root/v5.34.09
```

インストール作業

```
$ cd root
$ sudo ./configure --prefix=/usr/local/hep/root/v5.34.09
```

以下のコメントが出てくると configure は成功

```
To build ROOT type:

make
make  install
```

指示に従い、make 及び make install を行う。コンパイルする。

```
$ make
```

以下のコメントが出てくると make は成功

```
============================================================
===                ROOT BUILD SUCCESSFUL.                ===
=== Run 'make install' now.                              ===
============================================================
```

インストールする。

```
$ su
Password:
$ make install
...
$ exit
```

一時ファイルの削除

```
$ cd ../
$ pwd
/tmp
$ rm -rf root
```

環境変数ファイルの作成

ROOT 用の環境変数定義を書き込んだ setup.sh を準備して、ホームディレクトリに置く。

setup.sh

```
export ROOTSYS=/usr/local/hep/root/v5.34.09
export PATH=${ROOTSYS}/bin:${PATH}
export LD_LIBRARY_PATH=${ROOTSYS}/lib/root:${LD_LIBRARY_PATH}
```

ホームディレクトリ内のファイル.bash_profile に以下の一文を追加する。

```
source /usr/local/hep/root/setup.sh
```

その後、

```
source .bash_profile
```

2.2 ROOT をコマンドラインで使う

2.2.1 ROOT を起動する

```
$ root
```

ROOT の起動画面が立ち上がり、

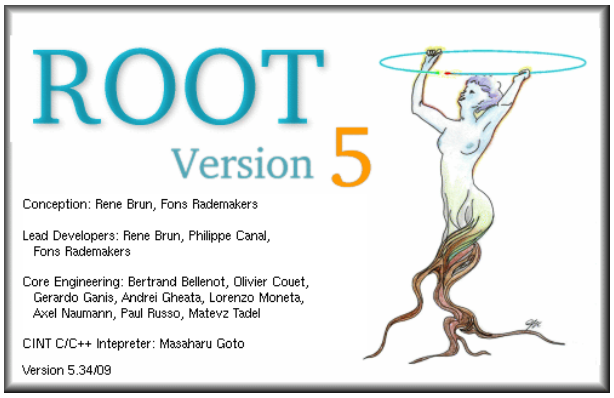


Figure 2.1: ROOT の起動画面

```
****************************************
*                                      *
*        W E L C O M E  to  R O O T    *
*                                      *
*   Version   5.34/09      26 June 2013  *
*                                      *
*  You are welcome to visit our Web site *
*          http://root.cern.ch         *
*                                      *
****************************************

ROOT 5.34/09 (v5-34-09@v5-34-09, Jun 26 2013, 17:10:36 on macosx64)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
```

という一連の情報が表示された後、

```
root[0]
```

となって ROOT が立ち上がり操作可能になる。

ROOT の起動画面を省略する

root 起動時に-l オプションをつければ起動画面は省略される。

```
$ root -l
```

ROOT の起動画面を常に省略する

いちいち -l オプションをつけるのが面倒くさい人は bash 起動時に自動的に読み込まれるファイル ~/.bash_profile に次の文章を追加する。

```
alias root="root -l"
```

次に source コマンドを使用すればよい。

```
$ source ~/.bash_profile
```

この状態でも ROOT の起動画面を省略しない場合には

```
$ \root
```

ROOT がデフォルトで読み込むファイル

2.2.2 ROOT を終了する

```
root[] .q
```

2.3 ROOT でマクロを読む/実行する方法

ROOT への命令文が書かれたプログラムのソースコード hello.cpp を準備しよう。

hello.cpp

```
#include <iostream>
void hello(){
  std::cout << "Hello, world" << std::endl ;
}
```

ざっとプログラムの流れを眺める。

1. #include <iostream>
   #include とは、C++のお作法であってヘッダファイル/ライブラリを読み込むということである。今の場合 C++で標準入出力を行う為に必要となるライブラリ<iostream>を読み込むという意味

2. void hello(){
   マクロの名前を準備するかを記述している箇所である。C++では関数を定義する時に
   ＜型名＞ ＜関数名＞ (＜引数＞){HogeHoge}
   という書き方をする。今の場合だと
   ＜型名＞ = void、＜関数名＞=hello、＜引数＞=無し、といった具合である。暫くの間はマクロ名とファイル名を同じにする。

3. std::cout << "Hello, world" << std::endl ;
   std::cout は標準出力を意味している。<<は標準出力先に続き文字列や数字を渡す。"Hello, world"を標準出力に渡している。std::endl は改行を意味する。

4. }
   2 行目の{に対応する括弧

今はソースコードを理解しなくてもいいが、気になる人は付録 A.1 などを参考にして理解せよ。

2.3.1 hello.cpp の実行方法 1

hello.cpp を ROOT 起動時に実行するには、

```
$ root hello.cpp
```

上記のコマンドを実行すると、

```
root [0]
Processing hello.cpp...
Hello, World
```

2.3.2 hello.cpp の実行方法 2

別の方法は ROOT をいったん起動して、プログラムを”ロード”して実行するという手段。

```
$ root
root [0] .L hello.cpp
root [1] hello()
Hello, World
```

2.3.3 hello.cpp の実行方法 3(推奨)

ロードする時に C++コンパイラを通してマクロに含まれるエラーメッセージなどを表記してくれる方法。やり方は’+’ をロードするファイル名の末尾につける。(http://root.cern.ch/drupal/content/compiling-macros)

```
$ root
root [0] .L hello.cpp+
```

例えば hello.cpp の std::endl; のセミコロンが無い時にはどうなるかというと、下のようにエラーメッセージと何が悪いのかをコンパイラが返してくれる。

```
Info in <TUnixSystem::ACLiC>: creating shared library /Users/SATOROOT/hello_cpp.so
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.cxx:17:
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.h:34:
/Users/SATOROOT/hello.cpp:3:42: error: expected ';' after expression
std::cout << "Hello, World" <<std::endl
^
;
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.cxx:17:
In file included from /Users/SATOROOT/hello_cpp_ACLiC_dict.h:18:
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:971:7:  \
warning: private field 'type' is not used [-Wunused-private-field]
int type;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:972:7: \
warning: private field 'tagnum' is not used [-Wunused-private-field]
int tagnum;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:973:7: \
warning: private field 'typenum' is not used [-Wunused-private-field]
int typenum;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:975:19: warning: \
private field 'isconst' is not used
[-Wunused-private-field]
G__SIGNEDCHAR_T isconst;
^
/usr/local/hep/root/v5.34.09/include/root/G__ci.h:977:29: warning: \
private field 'dummyForCint7' is not used
[-Wunused-private-field]
struct G__DUMMY_FOR_CINT7 dummyForCint7;
^
5 warnings and 1 error generated.
clang: error: no such file or directory: '/Users/SATOROOT/hello_cpp_ACLiC_dict.o'
Error in <ACLiC>: Compilation failed!
```

2.4 幾つかの常套手段

2.4.1 引数を数字にする

引数としてある整数を与えて、その数自身、その数の二乗、その数の三乗を出力するサンプルプログラムが usual1.cpp である。

usual1.cpp

```
#include <iostream>
#include "TMath.h"
void usual1(int i){
  std::cout << i << std::endl ;
  std::cout << TMath::Power(i,2) << std::endl ;
  std::cout << TMath::Power(i,3) << std::endl ;
}
```

実行方法は次の通りである。

```
$ root
root [0] .L usual1.cpp+
root [1] usual1(3)
3
9
27
```

なお、#include "TMath.h"という命令文によって、ROOT に組み込まれている定数や数式演算を使用可能にしている。
http://root.cern.ch/root/html/TMath.html

2.4.2 引数を文字列にする

引数を文字列にしたい場合には

usualchar.cpp

```
#include <iostream>
void usualchar(char *name){
  std::cout << "character string = " << name  << std::endl ;
}
```

実行方法は次の通りである。

```
$ root
root [0] .L usualchar.cpp
root [1] usualchar("test")
character string = test
```

2.4.3 引数を複数にする

複数個の引数を与えたい時には引数の () の中に、, を挟んで定義すれば良い。

usualage.cpp

```
#include <iostream>
void usualage(char *name,int i){
  std::cout << name << " is " << i << " years old." << std::endl ;
}
```

実行方法は次の通りである。

```
$ root
root [0] .L usualage.cpp
root [1] usualage("satoru",24)
satoru is 24 years old.
```

2.5 関数を描く

早速、関数を ROOT で関数を描こう。sinfunction.cpp を見てほしい。

sinfunction.cpp

```
#include "TF1.h"
#include "TMath.h"
TF1 *sinfunction(){
  TF1 *f = new TF1("f","TMath::Sin(x)") ;
  f->Draw() ;
  return f ;
}
```

そしてとりあえず実行して欲しい。

```
$ root
root [0] .L sinfunction.cpp+
root [1] sinfunction()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(class TF1*)0x7fe64437b580
```

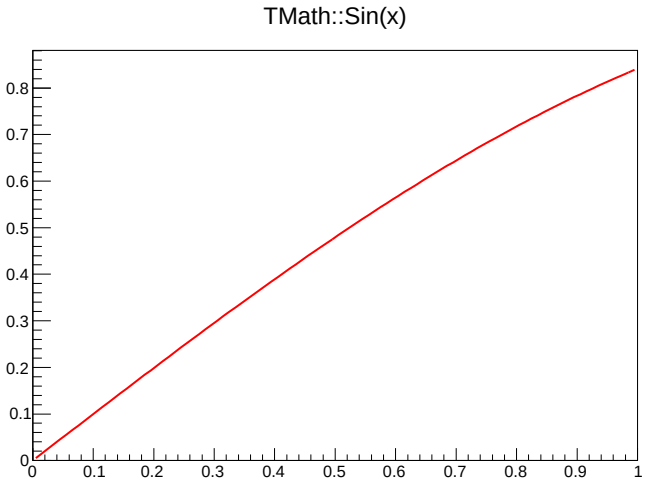


Figure 2.2: sinfunction.cpp の実行結果

おそらく、思っていたような定義域でない、軸に名前がついていないなどいろいろな不満があるだろう。それらは全て人間が指定してあげる必要がある。そういった手法も紹介していく。

これまであまりにすっ飛ばしてきたので、sinfunction.cpp を一行目から簡単に見ていこう。(C++のお作法をしっかり学ぶつもりは無いので説明はそれなりの質なので気になる箇所は自分でググるべし)

1. #include "TF1.h"
   ROOT で関数を描く為に必要となるライブラリ"TF1.h"を読み込むという意味

2. #include "TMath.h"
   ROOT で特定の定数 π や e などの値や、関数 $\sin(x)$、$\exp(x)$ などを使用する時に必要となるライブラリ"TMath.h"を読み込む。

3. TF1 *sinfunction(){
   マクロの名前を準備するかを記述している箇所である。＜型名＞ = TF1、＜関数名＞=sinfunction、＜引数＞=無し、を表している。*sinfunction というのは sinfunction という関数のポインタを意味するが、これ以上は触れない。

4. `TF1 *f = new TF1("f","TMath::Sin(x)") ;`
   ここからが `sinfunction.cpp` の本文。この行の左辺では`"TF1"`という型のポインタ `f` を定義している。`new` とは C++のお作法でメモリを動的に確保する為のものである。`TF1("f","TMath::Sin(x)")` とは左辺で定義した `f` というポインタの名前を `f` として、その間数は `TMath` というライブラリの中で定義された `Sin(x)` という関数にするという意味。

5. `f->Draw() ;`
   `f` というポインタを描く。4 行目のやり方で定義されたポインタ `f` に対して命令を与える時には`->`というアロー演算子を用いる。

6. `return f ;`

7. `}`
   3 行目の`{`に対応する括弧。

2.5.1 練習

1. 定義域を $-\pi$ から π に変更せよ。

   ヒント `http://root.cern.ch/root/html/TF1.html#TF1:TF1@1`

2. $2\sin(x/2)$ を描け。

   ヒント `http://root.cern.ch/root/html/TF1.html#TopOfPage` の B - Expression using variable x with parameters

   ヒント `http://root.cern.ch/root/html/TFormula.html#TFormula:SetParameter`

3. $\sin(x)$ と $\cos(x)$ を一緒に描け。また $\sin(x)$ の線を赤色、$\cos(x)$ の線を緑色にせよ。

   ヒント `http://root.cern.ch/root/html/TF1.html#TF1:Draw`

   ヒント `http://root.cern.ch/root/html/TAttLine.html#TAttLine:SetLineColor`

2.5.2 解答例

1. 定義域を $-\pi$ から π に変更せよ。

sinfunctionsol1.cpp

```
...
TF1 *sinfunctionsol1(){
  double pi = TMath::Pi() ;

  TF1 *f = new TF1("f","TMath::Sin(x)", -pi, pi) ;
  ...
}
```

2. $2\sin(x/2)$ を描け。

sinfunctionsol2.cpp

```
...
TF1 *sinfunctionsol2(){
  double pi = TMath::Pi() ;

  TF1 *f = new TF1("f","[0]*TMath::Sin([1]*x)", -pi, pi) ;
  f->SetParameter(0, 2.) ;
  f->SetParameter(1, 0.5) ;
  ...
}
```

3. sin(x) と cos(x) を一緒に描け。また sin(x) の線を赤色、cos(x) の線を緑色にせよ。

sinfunctionsol3.cpp

```
#include "TF1.h"
#include "TMath.h"
TF1 *sinfunctionsol3(){
  double pi = TMath::Pi() ;

  TF1 *f = new TF1("f","TMath::Sin(x)", -pi, pi) ;
  TF1 *f2 = new TF1("f2","TMath::Cos(x)", -pi, pi) ;

  f->SetLineColor(kRed) ;
  f2->SetLineColor(kGreen) ;

  f->Draw() ;
  f2->Draw("same") ;
  return f ;
}
```

2.6 ヒストグラムを描く

hist1.cpp

```
#include "TH1.h"
#include <iostream>

TH1D *hist1(){
  std::cout << "Start!!" << std::endl;
  TH1D *h = new TH1D("h","h",100,-5.,5.);
  h->FillRandom("gaus") ;
  h->Draw();
  return h;
}
```

まずは実行してみてほしい。すると、

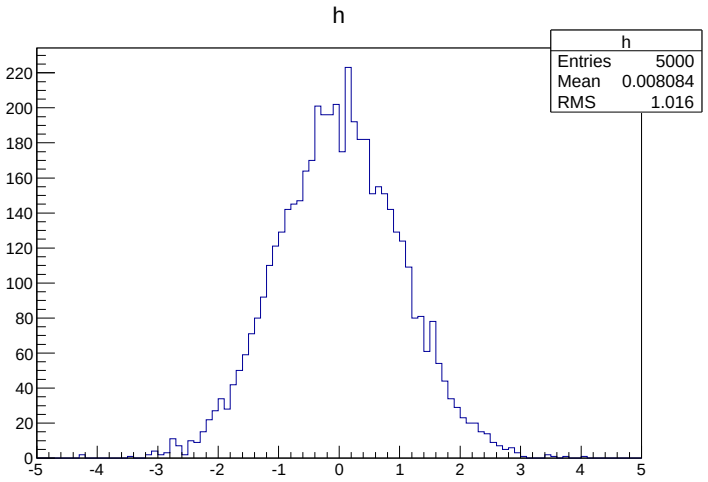


Figure 2.3: ヒストグラムの図

2.6.1 練習

1. ヒストグラムを統計誤差付きで評価せよ

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:Draw

   ヒント http://root.cern.ch/root/html/THistPainter.html#HP01a

   ヒント http://root.cern.ch/root/html/THistPainter.html#HP01b

2. ヒストグラムの最大値を取得せよ。

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetMaximum

3. ヒストグラムの最大値が納められた bin の bin 番号を取得せよ。

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetMaximumBin

4. ヒストグラムの最大値が納められた bin のエラーの値を取得せよ。

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetBinError

5. GUI を用いてグリッドを描け。最終的に図 2.4 のように描け。

   ヒント GUI では＜ View ＞から＜ Editor ＞を選択する。するとキャンバスの左側に様々な編集ツールが表示される。

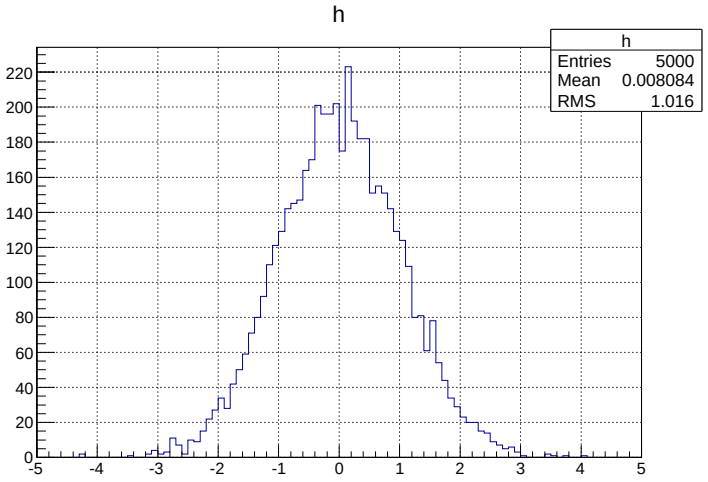


Figure 2.4: グリッドを描いたヒストグラム

6. ROOT で `h->Draw();` を行ったとき、
   `Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1`
   というメッセージが出ただろう。文字どおり、c1 というキャンバスが作られ、そこに h が描画されている。c1 を”c1.eps”として保存せよ。

   ヒント ROOT のキャンバスなどは全て `TObject` からの派生である。
   `http://root.cern.ch/root/html/TObject.html#TObject:SaveAs`

2.6.2 解答例

1. ヒストグラムを統計誤差付きで評価せよ

hist1sol1.cpp
```
...
TH1D *hist1sol1(){
  ...
  h->Draw("E");
  ...
}
```

2. ヒストグラムの最大値を取得せよ。

```
root [0] .L hist1.cpp+
root [1] hist1()
Start!!
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
(class TH1D*)0x7f97b4b7bda0
root [2] h->GetMaximum()
(const Double_t)2.23000000000000000e+02
```

3. ヒストグラムの最大値が納められた bin の bin 番号を取得せよ。

```
root [3] h->GetMaximumBin()
(const Int_t)52
```

4. ヒストグラムの最大値が納められた bin のエラーの値を取得せよ。

```
root [4] int maxbinnum = h->GetMaximumBin()
root [5] h->GetBinError(maxbinnum)
(const Double_t)1.49331845230680784e+01
```

5. GUI を用いてグリッドを描け。最終的に図 2.4 のように描け。

6. c1 を”c1.eps”として保存せよ。

```
root [6] c1->SaveAs("c1.eps")
Info in <TCanvas::Print>: eps file c1.eps has been created
```

2.7 TRandom と TCanvas

乱数とキャンバスの操作に出会う。

canran.cpp

```
#include "TCanvas.h"
#include "TH1.h"
#include "TRandom3.h"
#include "TStyle.h"
TCanvas *canran(){
  TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
  TRandom3 *r = new TRandom3();
  TH1D *h = new TH1D("h","h-title;x;y",100,-5,5) ;
  for(int i = 0 ; i<100000 ; i++){
    h->Fill(r->Uniform(-3.,3.)) ;
  }
  h->Draw("HE") ;
  c1->SaveAs("c1.eps") ;
  return c1 ;
}
```

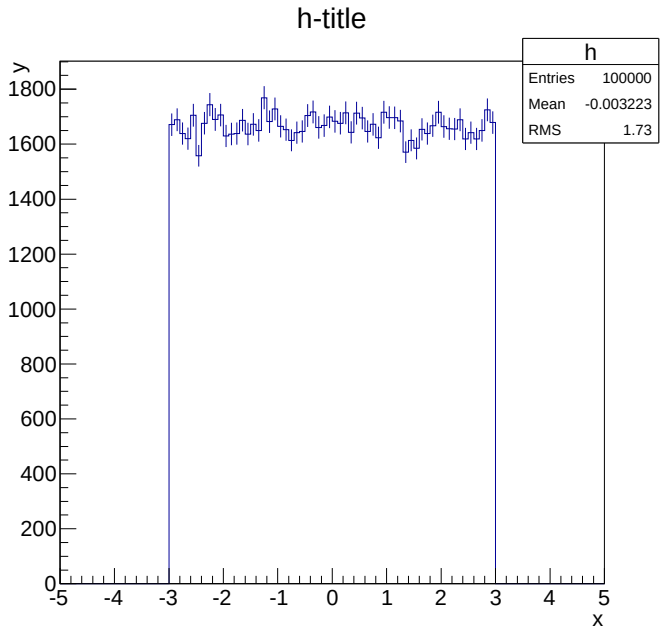


Figure 2.5: canran.cpp の実行結果

2.7.1 練習

1. プログラムの各行を説明せよ。

   ヒント メルセンヌツイスタとはメルセンヌ数という数の特徴を用いた乱数生成子のこと。

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html

   ヒント http://root.cern.ch/root/htmldoc/TRandom3.html

2. 今は h->Fill(r->Uniform(-1.,1.)) ; としているが、h->Fill(r->Exp(1.)) ;、h->Fill(r->Gaus(1.,1.)) ;、h->Fill(r->Binomial(3,0.3)) ;、h->Fill(r->PoissonD(1)) ; などとした時の挙動を確かめよ。

   ヒント これを気に幾つかの基本的な確率分布について調べよ。

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Exp

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Gaus

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Binomial

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Poisson

3. hist1 を繰り返し実行した時にヒストグラムに変化があるかどうか検証せよ。変化がない場合、この原因を突き止めて実行毎に違うヒストグラムが出来上がるような仕様へ変更せよ。

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:TRandom

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:SetSeed

2.7.2 解答例

1. プログラムの各行を説明せよ。

2. 今は h->Fill(r->Uniform(-1.,1.)) ; としているが、h->Fill(r->Exp(1.)) ;、h->Fill(r->Gaus(1.,1.)) ;、h->Fill(r->Binomial(3,0.3)) ;、h->Fill(r->PoissonD(1)) ; などとした時の挙動を確かめよ。

3. hist1 を繰り返し実行した時にヒストグラムに変化があるかどうか検証せよ。変化がない場合、この原因を突き止めて実行毎に違うヒストグラムが出来上がるような仕様へ変更せよ。

canransol1.cpp

```
...
TCanvas *canransol1(){
  TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
  TRandom3 *r = new TRandom3() ;
  r->SetSeed(unsigned (time(NULL))) ;
  ...
}
```

2.8 TGraph

ROOT を使って x 座標及び y 座標を与えてグラフを描くことが大いに考えられる。ROOT では TGraph を使うことで実現できる。

2.8.1 ファイルから読み込んでグラフ化する

シンチレータと光電子増倍管を組み合わせた検出器を考える。光電子増倍管からの信号にある閾値を設け、光電子増倍管に印加する電圧を変化させて単位時間あたりにどれだけの収量が得られたかを測定した実験を考える。この時、下記のような形式のデータファイルをきっと作ることだろう。

plateaudata.plt

```
# HV[V]  Counts[c/min]
1400  3
1450  1
1500  2
1550  10
1600  14
1650  20
1700  80
1750  250
1800  600
1850  900
1900  1000
1950  1050
2000  1060
2050  1090
2100  1095
2150  1100
2200  1105
2250  1108
2300  1109
2350  1105
2400  1110
2450  1120
2500  1150
2550  1240
2600  1350
2650  1505
2700  1700
```

plateaudata.plt をプロットする方法はいくつか在る。おそらく入り番簡単なのは gnuplot(http://www.gnuplot.info) を用いる方法である。gnuplot でこのテキストを図に起こす方法は下記の通り。

```
$ gnuplot
gnuplot> plot "plateaudata.plt"
```

実行結果は図 2.6 のようである。
これを ROOT で行う最低限のサンプルプログラムを示す。

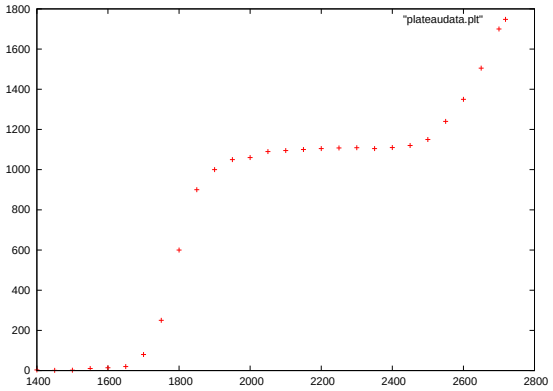


Figure 2.6: plateaudata.plt を gnuplot を用いて描いた時の様子。

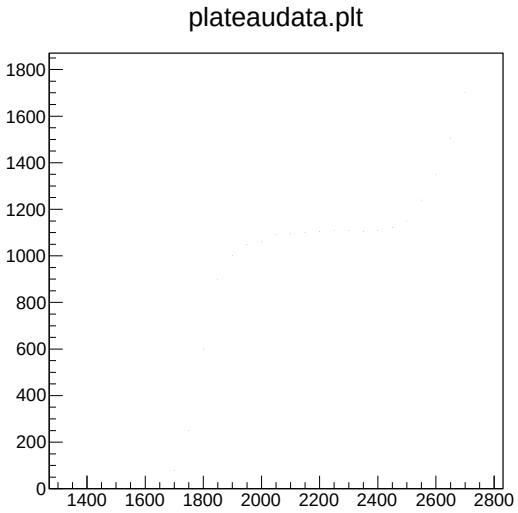


Figure 2.7: plateaudata.plt を plateauplot.cpp を用いて描いた時の様子。

plateauplot.cpp

```
#include "TGraph.h"
#include "TCanvas.h"

TCanvas *plateauplot(){
  TCanvas *c1 = new TCanvas("c1", "c1", 600, 600) ;
  TGraph *f = new TGraph("plateaudata.plt", "%lf %lf") ;
  f->Draw("AP") ;
  return c1 ;
}
```

この実行結果が図 2.7 である。

練習

1. プログラムの各行を理解せよ。

   ヒント http://root.cern.ch/root/html/TGraph.html

   ヒント http://root.cern.ch/root/html/TGraph.html#TGraph:TGraph@10

   ヒント http://root.cern.ch/root/html/TGraph.html#TGraph:Draw

   ヒント http://root.cern.ch/root/html/TGraphPainter.html

2. 引数をデータファイル名にするように変更せよ。

3. 図 2.7 を見てわかるように、デフォルトではマーカーのサイズが小さい。マーカーサイズを調整したり、軸のタイトルを変更するなどしておしゃれせよ。

   ヒント http://root.cern.ch/root/html/TAttMarker.html#TAttMarker:SetMarkerStyle

   ヒント http://root.cern.ch/root/html/TAttMarker.html#TAttMarker:SetMarkerSize

   ヒント http://root.cern.ch/root/html/TAttMarker.html#TAttMarker:SetMarkerColor

   ヒント http://root.cern.ch/root/html/TAttPad.html#TAttPad:SetLeftMargin

解答例

1. プログラムの各行を理解せよ。
2. 引数をデータファイル名にするように変更せよ。
3. 図 2.7 を見てわかるように、デフォルトではマーカーのサイズが小さい。マーカーサイズを調整したり、軸のタイトルを変更するなどしておしゃれせよ。

plateauplotsol1.cpp

```
...
TCanvas *plateauplotsol1(char *datafilename){
  TCanvas *c1 = new TCanvas("c1", "c1", 600, 600) ;
  c1->SetGrid(1,1) ;
  c1->SetLogy() ;
  c1->SetLeftMargin(0.14) ;
  TGraph *f = new TGraph(datafilename, "%lf %lf") ;
  f->SetMarkerStyle(20) ;
  f->SetMarkerSize(1.1) ;
  f->SetMarkerColor(kRed) ;
  f->SetTitle("Plateau Curve") ;
  f->GetXaxis()->SetTitle("Voltage[V]") ;
  f->GetYaxis()->SetTitle("Counts[c/min]") ;
  f->GetYaxis()->SetTitleOffset(1.4) ;
  f->Draw("AP") ;
  return c1 ;
}
```

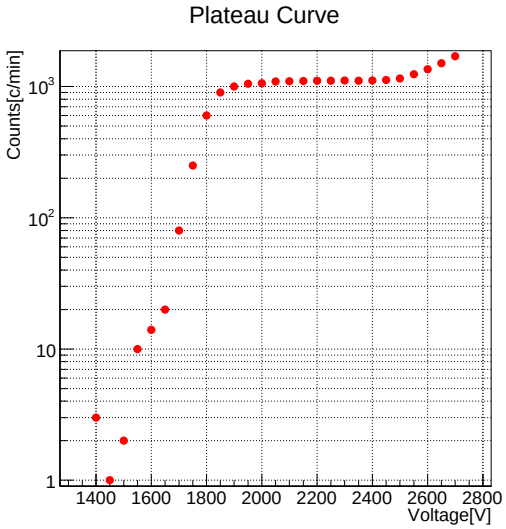


Figure 2.8: plateauplotsol1.cpp の実行結果。

2.8.2 ランダムウォーク

randomwalk.cpp

```
#include "TRandom3.h"
#include "TCanvas.h"
#include "TGraph.h"

TCanvas *randomwalk(){
  const int imax = 100 ;
  double x[imax] = {0.} ;
  double y[imax] = {0.} ;

  TRandom3 RandomGenerator(unsigned(time(NULL))) ;
  TCanvas *c1 = new TCanvas("c1", "c1", 600,600) ;
  c1->SetGrid(1,1) ;
  for(int i = 1 ; i < imax ;i++){
    x[i] = RandomGenerator.Uniform(-1., 1.) + x[i-1] ; // 前の位置から [-1,1] だけ x 座標を移動
    y[i] = RandomGenerator.Uniform(-1., 1.) + y[i-1] ; // 前の位置から [-1,1] だけ y 座標を移動
  }
  TGraph  *f  = new TGraph(imax, x, y) ;
  f->SetLineStyle(1) ;
  f->SetLineWidth(2) ;
  f->SetLineColor(kRed) ;
  f->SetMarkerStyle(20) ;
  f->SetMarkerSize(1.1) ;
  f->SetMarkerColor(kBlue) ;

  f->Draw("APL") ;
  return c1 ;
}
```

練習

1. プログラムの各行を説明せよ

   ヒント http://root.cern.ch/root/html/TGraph.html

   ヒント http://root.cern.ch/root/html/TGraph.html#TGraph:TGraph@2

2. 今は x,y それぞれの変化は [-1,1] で与えているが、現在位置から半径 1 の円周上が新しい点になるようにプログラムを変更せよ。これは TGraph の練習というよりも乱数の練習である。

   ヒント http://root.cern.ch/root/htmldoc/TRandom.html#TRandom:Circle

解答例

1. プログラムの各行を説明せよ
2. 今は x,y それぞれの変化は [-1,1] で与えているが、現在位置から半径 1 の円周上が新しい点になるようにプログラムを変更せよ。

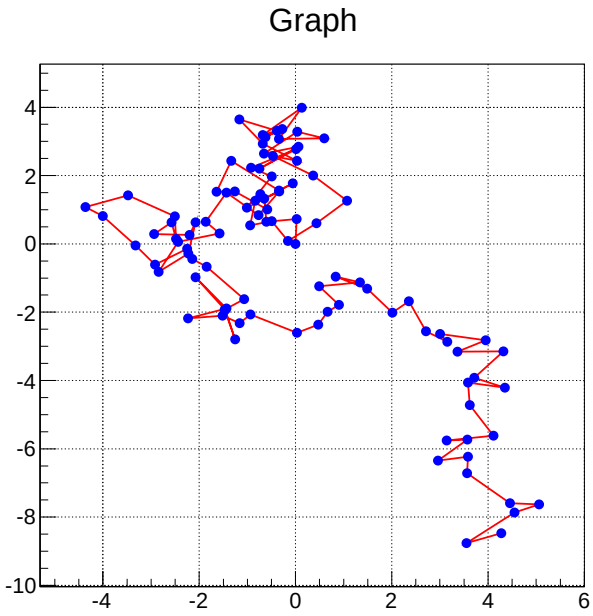


Figure 2.9: randomwalk.cpp の実行結果

randomwalksol1.cpp

```
...
TCanvas *randomwalksol1(){
  const int imax = 100 ;
  double x[imax] = {0.} ;
  double y[imax] = {0.} ;
  double deltax ;
  double deltay ;
  double radius = 1.0 ;
  ...
  for(int i = 1 ; i < imax ;i++){
    RandomGenerator.Circle(deltax, deltay, radius) ; // 半径 radius の円周上に一様に乱数を生
成する。
    x[i] = deltax + x[i-1]; // 前の座標から deltax だけ移動する
    y[i] = deltay + y[i-1]; // 前の座標から deltay だけ移動する
  }
  TGraph  *f  = new TGraph(imax, x, y) ;
  ...
}
```

念のため、実行方法は次の通り。

```
$ root
root[0] .L plateauplotsol1.cpp+
root[1] plateauplotsol1("plateaudata.plt")
```

2.9 任意の関数に従うヒストグラムを描く

自分で定義した関数に従って乱数を生成してヒストグラムを描こう。

ranfun.cpp

```
#include "TCanvas.h"
#include "TF1.h"
#include "TH1.h"
#include "TMath.h"
TH1D *ranfun(){
  double range_min = 0. ; // 0 [ns]
  double range_max = 8000.e-9 ;// 8000 [ns]
  double tau       = 2.2e-6 ; // it means lifetime
  double bgd = 0.5 ; // Background
  int nbin = 100 ; // histgram bin num
  int imax = 100000 ; // event

  TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
  c1->SetGrid(1,1) ; // Canvas c1 にグリッドを描く
  c1->SetLogy(1) ;  // Canvas c1 の縦軸を log で

  TF1 *f = new TF1("f","TMath::Exp(-x/[0])+[1]", range_min, range_max) ;
  f->SetParameter(0, tau) ;
  f->SetParameter(1, bgd) ;
  //  f->SetParameters(tau,bgd) ; // <--上の二行はこの一行と等価

  TH1D *h = new TH1D("h","Decay curve (Muon);TDC [s] ; Counts ", nbin, range_min, range_max) ;
  for(int i=0; i < imax; i++){
    h->Fill(f->GetRandom()) ;
  }
  h->Draw("HE") ;

  /*****************************************************************************\
  * Decay Curve Fitting
  \*****************************************************************************/
  /* If you want to use fit, then please uncomment
  gStyle->SetOptFit(1101) ;
  TF1 *muon = new TF1("muon","[0]*TMath::Exp(-x/[1])+[2]") ;
  muon->SetParameters(2e+3, 2e-6, 1e+2) ;
  muon->SetLineColor(kBlue) ;
  muon->SetLineWidth(4) ;
  h->Fit(muon) ;
  */
  return h ;
}
```

2.9.1 練習

1. プログラムの各行の役割を理解せよ。

   ヒント http://root.cern.ch/root/html/TF1.html#TF1:GetRandom

2. 図 2.10 のようなおしゃれをしたヒストグラムを描け。

   ヒント http://root.cern.ch/root/html/TH1.html#TH1:GetXaxis

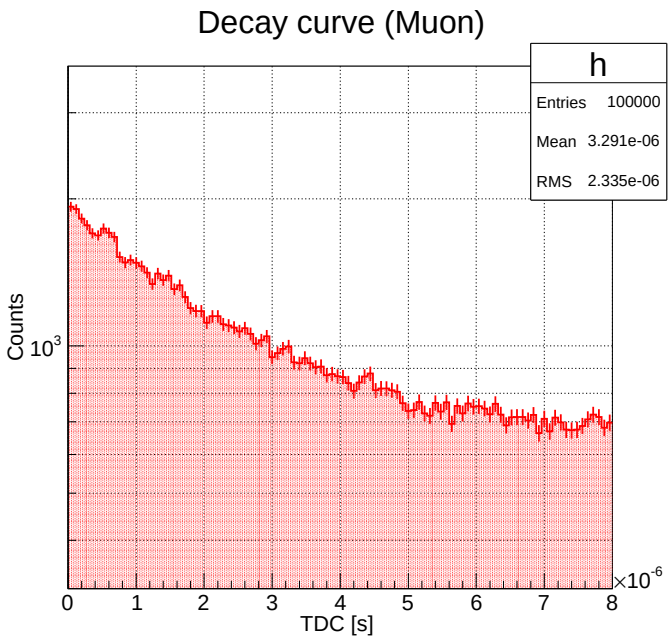


Figure 2.10: 各種の設定をいじったヒストグラム

ヒント http://root.cern.ch/root/htmldoc/TAxis.html#TAxis:CenterTitle

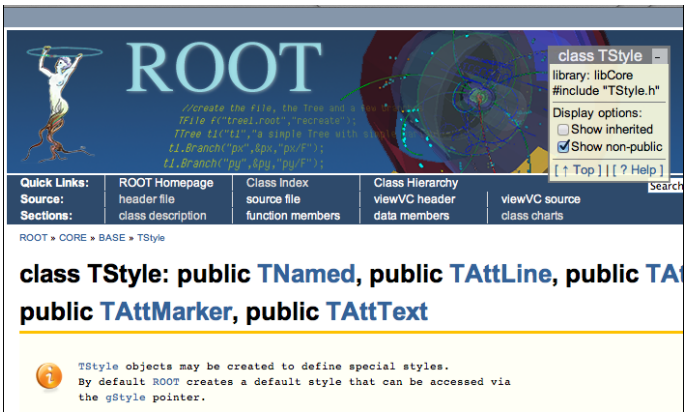
ヒント http://root.cern.ch/root/html/TH1.html#TH1:SetTitleOffset

ヒント http://root.cern.ch/root/html/TAttFill.html#TAttFill:SetFillColor

ヒント http://root.cern.ch/root/html/TAttFill.html#TAttFill:SetFillStyle

3. ranfun.cpp の下部 Decay Curve Fitting 以下のコメントアウトされている箇所/* ... */をアンコメントして実行してみよ。

   ヒント この状態で ranfun.cpp をコンパイルオプション付きでロードしたらエラーメッセージが出るだろう。何を include すべきかはコンパイラが認識し兼ねている単語と' ライブラリ' などの言葉と一緒に検索せよ。それが ROOT に関連しているものであれば各 class を説明しているページの右上に何を include すべきかが書いてある。
   http://root.cern.ch/root/html/TStyle.html



4. フィッティングの情報が誤差付きで描かれるように変更せよ。

   ヒント http://root.cern.ch/root/html/TStyle.html#TStyle:SetOptFit

2.9.2 解答例

1. プログラムの各行の役割を理解せよ。
2. 図 2.10 のようなおしゃれをしたヒストグラムを描け。

ranfunsol1.cpp

```
...
TH1D  *ranfunsol1(){
  ...
  h->GetXaxis()->CenterTitle() ;
  h->GetYaxis()->CenterTitle() ;
  h->GetYaxis()->SetTitleOffset(1.4) ;
  h->SetFillStyle(3002) ;
  h->SetFillColor(kRed-4) ;
  h->SetLineColor(kRed) ;
  h->SetLineWidth(2) ;
  ...
}
```

3. ranfun.cpp の下部 Decay Curve Fitting 以下のコメントアウトされている箇所/* ... */をアンコメントして実行してみよ。
4. フィッティングの情報が誤差付きで描かれるように変更せよ。

ranfunsol2.cpp

```
...
TH1D  *ranfunsol2(){
  ...
  // If you want to use fit, then please uncomment
  gStyle->SetOptFit(1111) ;
  TF1 *muon = new TF1("muon","[0]*TMath::Exp(-x/[1])+[2]") ;
  muon->SetParameters(2e+3, 2e-6, 1e+2) ;
  muon->SetLineColor(kBlue) ;
  muon->SetLineWidth(4) ;
  h->Fit(muon) ;
  ...
}
```

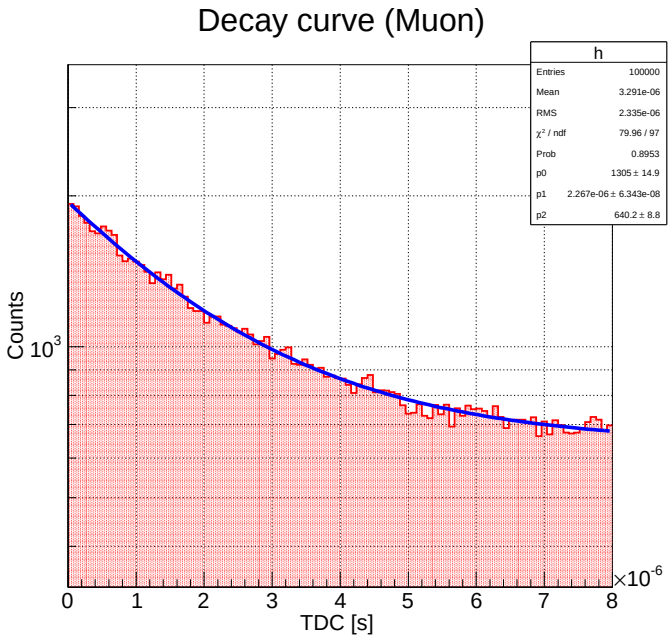


Figure 2.11: `ranfunsol2.cpp` の実行結果

2.10 File への出力

fileout.cpp

```
#include "TCanvas.h"
#include "TH1.h"
#include "TMath.h"
#include "TRandom3.h"
#include <fstream>
#include <iostream>
TCanvas *fileout(){
  std::ofstream outputfile; // file の出力先
  outputfile.open("output.plt" );// file を開く

  TRandom3 Random(unsigned(time(NULL)));

  TCanvas *c1 = new TCanvas("c1","c1") ;

  int imax = 100000; // event 数の最大値
  double start ; // start の時刻を擬似的に与える。
  double stop  ; // stop の時刻を擬似的に与える。
  double tdc   ; // TDC でのカウント数を擬似的に与える。
  double life = 2.2e-6 ; // Muon の平均寿命を入力する。

  TH1D *h1 = new TH1D("h1","h1",200,0,8000e-9) ; // [s]
  for(int i = 0; i < imax; i++){
    start = Random.Uniform(0., 1.e-9) ; // start の時刻を擬似的に与える。
    stop = Random.Exp(life) ; // 指数関数に従った乱数だけ立った時刻を stop とする
    stop += start ; // start してから 指数関数に従った乱数だけ立った時刻を stop とする
    tdc = stop - start ; // このプログラムだけを見るとこの処理は余分な処理だが、TDC の理の為
    std::cout << tdc << " [s] :: " << start << "" << stop << std::endl;
    // output という出力先に start と stop をスペース区切りで出力する
    outputfile << start << " " << stop << " " << endl ;
    h1->Fill(tdc) ; // ヒストグラムに詰める
  }
  h1->Draw("H") ; // 確認用にヒストグラムを出力する。
  outputfile.close(); // ファイルを閉じる return c1 ;
}
```

output.plt

```
4.8318e-10 1.59013e-06
8.88674e-11 2.45466e-06
8.04413e-10 1.65805e-06
...
```

2.10.1 練習

1. プログラムの各行を理解せよ

   ヒント <fstream> なるものを include している。これはファイル操作の時に今回使用するライブラリである。(今回は使用していないが、ROOT には TFile.h なるクラスも存在する。)

2. 出力先のファイル名を今はマクロ内に直書きしているが、引数をファイル名にせよ

2.10.2 解答例

1. プログラムの各行を理解せよ
2. 出力先のファイル名を今はマクロ内に直書きしているが、引数をファイル名にせよ

fileoutsol1.cpp

```
...
TCanvas *fileoutsol1(char *outputfilename){
  std::ofstream outputfile; // file の出力先
  outputfile.open(outputfilename );// file を開く
  ...
}
```

```
root[0] .L fileoutsol1.cpp+
root[1] fileoutsol1("output2.plt")
```

2.11 File からの入力

先程出力したファイルからデータを読み込んでヒストグラムを描くことを経験しよう。

filein.cpp

```
#include "TCanvas.h"
#include "TF1.h"
#include "TH1.h"
#include "TMath.h"
#include "TStyle.h"
#include <fstream>
TCanvas *filein(char *file_name){
  double range_min = 0. ;
  double range_max = 8000.e-9 ;// 8000 [ns]
  int nbin = 100 ;
  double start ; // TDC start
  double stop  ; // TDC stop
  double tdc   ; // delta T

  TCanvas *c1 = new TCanvas("c1","c1",600,600) ;
  c1->SetGrid(1,1); // Canvas c1 にグリッドを描く
  c1->SetLogy(1) ;  // Canvas c1 の縦軸を log で
  gStyle->SetOptFit(1) ;
  TH1D *h = new TH1D(file_name,"Decay curve (Muon);TDC [s] ; Counts ",
  nbin, range_min, range_max);
  ifstream fin(file_name) ;
  while(fin >> start >> stop){
    tdc = stop - start ;
    h -> Fill(tdc) ;
  }
  h->Draw("HE");

  /****************************************************************************** \
  * Decay Curve Fitting
  \******************************************************************************/
  TF1 *muon = new TF1("muon","[0]*(TMath::Exp(-x/[1])+[2])") ;
  muon->SetParameters(2e+3, 2e-6, 0.5) ;
  muon->SetLineColor(kBlue) ;
  muon->SetLineWidth(4) ;
  h->Fit(muon) ;

  return c1 ;
}
```

2.12 tree に出会う

ROOT には拡張子に.root を拡張子としたファイルがある。(以下 root ファイルと呼ぶ。) ここでは root ファイルにデータを詰める方法とその使い方を示す。やることは

1. start と stop の 2 行が書かれたデータを開く
2. start と stop、及びその差を読んで TDC の値とする。
3. start、stop、TDC の値を Tree というものに格納する。
   http://root.cern.ch/drupal/content/ttree-and-its-data
4. Tree を root ファイルに書き出す。

meettree.cpp

```
#include <fstream>
#include "TFile.h"
#include "TTree.h"

TTree *meettree(char *datafile, char *rootfile = "output.root"){

  double start ; // TDC start
  double stop  ; // TDC stop
  double tdc   ; // delta T

  TTree *tree = new TTree("tree","tree");  //TTree 作成
  //Branch 準備
  tree->Branch( "start", &start, "start/D" ); // start を格納する為のブランチ
  tree->Branch( "stop" , &stop , "stop/D" );  // stop を格納する為のブランチ
  tree->Branch( "tdc"  , &tdc  , "tdc/D" );   // start と stop の時間差 を格納する為のブランチ

  ifstream fin(datafile) ;
  while(fin >> start >> stop){
    tdc = stop - start ;
    tree->Fill() ;
  }

  TFile *fout = new TFile(rootfile, "recreate");
  tree->Write(); // tree を書き込む
  fout->Close(); // file close

  return tree ;
}
```

2.12.1 meettree.cpp を実行する

```
$ root
root [0] .L meettree.cpp+
root [1] meettree("output.plt","test.root")
(class TTree*)0x7fc1d22a1b70
```

すると、作業ディレクトリに test.root というファイルが出来ている。(出力ファイルのデフォルト名は output.root なので、meettree 実行時に第二引数を指定しなかった場合、出来上がる root ファイルは output.root である。) これが root ファイルである。

2.12.2 Tree を扱う

以降、前節で出力したファイル名が test.root だとして話を進める。test.root に収められている Tree の扱いの走りを紹介する。

```
$ root  test.root
root [0]
Attaching file test.root as _file0...
```

次に、今我々が扱えるものが何かを表示する。

```
root[1] .ls
TFile** test.root
TFile* test.root
KEY: TTree tree;1 tree
```

tree というのが存在するのがわかる。tree の情報を表示するには、Print を使う。

```
root [2] tree->Print()
******************************************************************************
*Tree    :tree      : tree                                                   *
*Entries :   100000 : Total =         4808328 bytes  File  Size =    2406559 *
*        :          : Tree compression factor =   2.00                       *
******************************************************************************
*Br    0 :start     : start/D                                                *
*Entries :   100000 : Total  Size=    1602694 bytes  File Size  =     801872 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   2.00     *
*.............................................................................*
*Br    1 :stop      : stop/D                                                 *
*Entries :   100000 : Total  Size=    1602664 bytes  File Size  =     801846 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   2.00     *
*.............................................................................*
*Br    2 :tdc       : tdc/D                                                  *
*Entries :   100000 : Total  Size=    1602634 bytes  File Size  =     801820 *
*Baskets :       26 : Basket Size=      32000 bytes  Compression=   2.00     *
*.............................................................................*
```

これらが tree で扱える情報である。

2.12.3 Tree からヒストグラムを描く

tree に入った情報を書き出すのは簡単である。

```
root [3] tree->Draw("start")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [4] tree->Draw("stop")
```

などとすれば、ヒストグラムが描かれる。(このテキストの fileout.cpp でどのような乱数で start や stop を与えたのかを思い出せ。)

何もしなければ bin 数やヒストグラムの領域は自動で決まるが設定することももちろん出来る。

```
root [5] tree->Draw("start>>h(100, 0., 1e-9)")
```

などとすればわかるだろう。

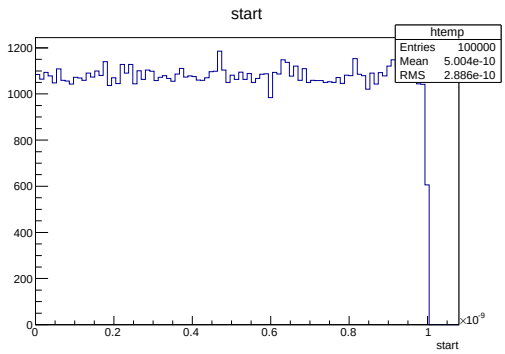


Figure 2.12: `tree->Draw("start")` 実行によって表示される `start` のヒストグラム

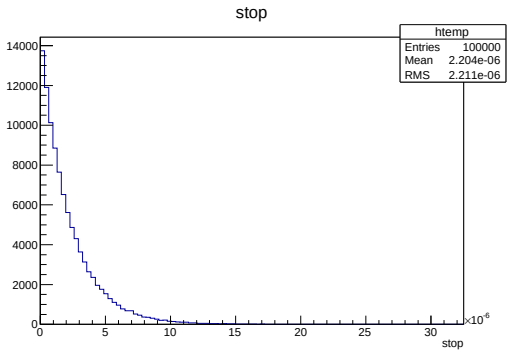


Figure 2.13: `tree->Draw("stop")` 実行によって表示される `stop` のヒストグラム

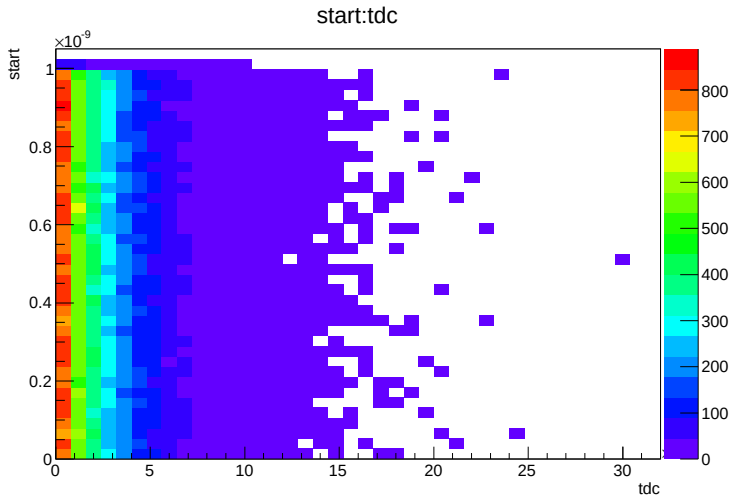


Figure 2.14: `start` 実行によって表示される `start` のヒストグラム

2.12.4 練習

1. コマンドライン上でで tdc の値格納用のヒストグラムのを用意した後、TDC のヒストグラムを描け。

   ヒント http://root.cern.ch/root/html/TTree.html#TTree:Draw@2

2. tdc を x 軸、start を y 軸とした図 2.14 のような２次元ヒストグラムをコマンドラインから描け。

   ヒント http://root.cern.ch/root/html/THistPainter.html#HP01c

2.12.5 解答例

1. コマンドライン上でで tdc の値格納用のヒストグラムのを用意した後、TDC のヒストグラムを描け。

```
root [] TH1D *h = new TH1D("h", "h", 100, 0., 8000e-9)
root [] tree->Draw("tdc>>h")
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

2. tdc を x 軸、start を y 軸とした図 2.14 のような２次元ヒストグラムをコマンドラインから描け。

```
root [] tree->Draw("start:tdc","","colz")
```

2.13 Tree から読んで描く

meettree2.cpp

```
#include "TCanvas.h"
#include "TFile.h"
#include "TH1D.h"
#include "TTree.h"

TH1D *meettree2(char *InputRootFileName){

  TCanvas *c1 = new TCanvas("c1", "c1") ;
  TFile *file = new TFile(InputRootFileName,"READ") ;
  TTree *t = (TTree*)file->Get("tree") ;

  TH1D *h = new TH1D("h","TDC",100, 0., 8000e-9) ; // 8000[ns]
  t->Draw("tdc>>h") ;

  c1->cd() ;
  h->Draw() ;

  return h ;
}
```

2.13.1 練習

1. プログラムの挙動を理解せよ。

   ヒント http://root.cern.ch/root/html/TFile.html#TFile:TFile@2

   ヒント http://root.cern.ch/root/html/TDirectoryFile.html#TDirectoryFile:Get

2. これまでの知識を動員して、meettree2.cpp を改良せよ。具体的には軸に単位を追加したり、自動的に Fit したりするなどせよ。

2.13.2 解答例

Chapter 3

ネイティブプログラミング

3.1 ネイティブプログラムへの移行準備

3.1.1 プログラミング言語とコンパイラ

とりあえず動かすことを目的としてきたので、一旦落ち着いてコンパイラの話をする。

今まで我々が扱ってきたソースコードは全て人間のための言葉であり、機械のための言葉ではない。このような言語を高水準言語と呼ぶ。一方、機会が実際に実行できる仕様になっている言語のことを低水準言語または機械語などと言ったりする。プログラマが書いた高水準言語を低水準言語に変換してくれるのがコンパイラである。

3.1.2 コンパイラ c++

C++用のコンパイラ。本テキストでは コンパイラとして c++を常に用いる。

3.1.3 コンパイルする

サンプルプログラム hello2.cpp をコンパイルする方法を紹介する。

hello2.cpp

```
#include <iostream>
int main(int argc, char** argv){
  std::cout << "Hello, World" << std::endl;
  return EXIT_SUCCESS ;
}
```

一番シンプルなコンパイル

```
$ c++ hello2.cpp
```

これで a.out が出来上がる。この a.out が実行ファイルである。実行ファイルを実行する方法は

```
$ ./a.out
Hello, World
```

実行ファイルの名前を指定してコンパイル

実行ファイル名を a.out 以外にしたい時には-o オプションを用いる。

```
c++ hello2.cpp -o hello2
```

これで実行ファイル hello2 が生成される。実行方法は先の場合とファイル名が違うだけで

```
$ ./hello2
Hello, World
```

全ての警告文を表記するオプションを追加してコンパイル

コンパイル時にオプション-Wall オプションを用いることで、コンパイラが警告できる警告文を全て表記させる事ができる。

```
c++ -Wall hello2.cpp -o hello2
```

と言った具合である。
　試しにサンプルプログラム hello2wrong.cpp

hello2wrong.cpp

```
#include <iostream>
int main(int argc, char** argv){
  std::cout << "Hello, World" << std::endl
  return EXIT_SUCCESS ;
}
```

をコンパイルしてみると、下記のようなエラー文を返してくれる。

```
$c++ -Wall hello2wrong.cpp -o hello2wrong
hello2wrong.cpp: In function ‘int main(int, char**)’:
hello2wrong.cpp:4: error: expected ‘;’ before ‘return’
```

3.1.4 オブジェクトファイル

初心者には更に厄介な事情がある。今までは直接実行ファイルを作成していたが、”関連してくるファイルが増えてきた時には、各ソースコードからオブジェクトファイルと呼ばれるものを作成し、オブジェクトファイルの組み合わせによって実行ファイルを作成する。” というプロセスを得るのが普通である。オプション-c をつけることによって、オブジェクトファイルを作成するところまでを行わせることが出来る。

3.2 Makefile その 1

3.2.1 Make(メイク)

メイクとは Makefile と呼ばれるルールを記述したファイル、または予め定義された規則などから UNIX の Shell が実行するコマンド列を作成するコマンドジェネレータの役目を持つ。メイクはファイル間の依存関係を整理するのにも使われ、大規模プログラミングでは必須のツールである。make というコマンドがまず自分の PC に導入されているかを確認しておいて欲しい。なければどこぞからインストールしてくること。make があるかどうかの確認は

```
$ which make
/usr/bin/make
```

make のヴァージョン確認は

```
$ make -v
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
```

```
PARTICULAR PURPOSE.

This program built for i386-apple-darwin11.3.0
```

3.2.2 Makefile

Makefile メイクで実行するコマンド列を定義しておくファイル名である。Makefile が存在するディレクトリ下において、

```
$ make
```

と入力することで Makefile にかかれた内容が実行される。なお、make で実行したいファイル名が Makefile2 の場合には

```
$ make -f Makefile2
```

make で実行したいファイル名が hogex の場合には

```
$ make -f hogex
```

などとすればよい。

3.2.3 直打ちメイクファイル

まずは hello2.cpp を make することで実行ファイルを作成する Makefile を作ってみよう。お作法を知らなくてもまずは真似をして書いてみて欲しい。

Makefile

```
# Makefile の中では #で始まる行は実際にはコメントアウトされます。
# hello2.cpp から hello2 という実行ファイルを作成する為の Makefile
all:hello2

hello2:
	c++ -Wall hello2.cpp -o hello2
```

ただし、'c++ -Wall hello2.cpp -o hello2' の行の行頭は<TAB>を用いること。
hello2.cpp と Makefile が同じディレクトリにいることを確認した後、早速 make を実行し、実行ファイル hello2 が出来上がっていることを確認しよう。

```
$ ls
Makefile hello2.cpp
$ make
c++     hello2.cpp   -o hello2
$ ls
Makefile hello2 hello2.cpp
```

直打ちメイクファイルの解説

Makefile の中身を順序立てて追っていく。

1. all:hello2
   コロン記号:の直前の all のことをターゲットという。make を実行すると、デフォルトで all というターゲットが存在するかどうかをチェックする。そして今は all が定義されているので、この行が処理の対象となる。次にターゲットの右にリストされるターゲットから実行する。今の場合、hello2 がリストされているので、次のターゲットが hello2 ということになる。

2. hello2:
   ターゲットが hello2 を意味している。この行はターゲットの場所を顕に示しただけであり、すぐ次の文へと移る。

3. 	c++ -Wall hello2.cpp -o hello2
   ターゲットが hello2 に属している実行分である。Makefile では実行分の行頭を<TAB>でインデントする必要があるので注意すること。この文で実行される内容は、hello2.cpp というソースコードをコンパイルして、実行ファイル hello2 を作るという動作である。

3.2.4 マクロを利用したメイクファイル

直打ちメイクファイルはかっこ悪いので少しずつランクアップをしよう。Makefile ではマクロという変数を用いることで、直打ちから脱却が出来る。その例が Makefike2 である。

Makefile2

```
# hello2.cpp から hello2 という実行ファイルを作成する為の Makefile
TARGET = hello2
COM    = c++
CFLAGS = -Wall
all:$(TARGET)

$(TARGET):
        $(COM) $(CFLAGS) $(TARGET).cpp -o $(TARGET)
```

マクロ = 値でマクロを定義してマクロに値を格納し、$(マクロ名) でマクロに格納された値を用いる。
make を実行し、実行ファイル hello2 が出来上がっていることを確認しよう。

```
$ ls
Makefile Makefile2 hello2.cpp
$ make -f Makefile2
c++ -Wall hello2.cpp -o hello2
$ ls
Makefile Makefile2 hello2 hello2.cpp
```

make の実行文が全てマクロが展開された形で行われていることが見てとれる。

3.2.5 オブジェクトファイルを意識したメイクファイル

後々の事を考えてオブジェクトファイルを意識した Makefile の例を示す。

Makefile3

```
# hello2.cpp から hello2 という実行ファイルを作成する為の Makefile
TARGET = hello2
COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o

all:$(TARGET)

$(TARGET):$(OBJS)
        $(COM) $(CFLAGS) $(TARGET).cpp -o $(TARGET)
$(OBJS):$(TARGET).cpp
        $(COM) -c $(TARGET).cpp
```

3.2.6 内部マクロを用いたメイクファイル

Makefile 内部では、内部マクロと言われる特殊な指定子を用いることが出来る。

記号	説明
$@	: ターゲットファイル名
$<	: 最初の依存ファイル名
$?	: ターゲットより新しい全ての依存ファイル名
$^	: 全ての依存ファイル名
$+	: Makefile と同じ順番の依存ファイル名
$*	: suffix を除いたターゲット名
$%	: アーカイブだった時のターゲットメンバ名

内部マクロを用いた Makefile の例を示す。

Maefile4

```
# hello2.cpp から hello2 という実行ファイルを作成する為の Makefile
TARGET = hello2
COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o

all:$(TARGET)

$(TARGET):$(OBJS)
        $(COM) $(CFLAGS) $(TARGET).cpp -o $^
$(OBJS):$(TARGET).cpp
        $(COM) -c $<
```

3.2.7 ターゲット clean

Makefile でよく使われるターゲットに clean というのがある。clean の役割はオブジェクトファイルを削除して全てのファイルを再コンパイル可能にする為のもの。

Maefile5

```
# hello2.cpp から hello2 という実行ファイルを作成する為の Makefile
TARGET = hello2
COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o

all:$(TARGET)
$(TARGET):$(OBJS)
        $(COM) $(CFLAGS) $(TARGET).cpp -o $^
$(OBJS):$(TARGET).cpp
        $(COM) -c $<
clean:
        rm -f *.o
```

Makefile4 と Makefile5 の違いは最後の二行だけである。使い方は

```
$ make -f Makefile5 clean
rm -f *.o
```

3.3 Makefile その 2:ファイルを分割する

整数の 3 乗を計算するプログラム cube.cpp を分割することを考える。

cube.cpp

```
#include "iostream"
int cube(int a) ;

int main(int argc, char** argv){
  std::cout << cube(4) << std::endl ;
  return EXIT_SUCCESS ;
}

int cube(int a){
  return a*a*a ;
}
```

この時のメイクファイルは

Makefile

```
TARGET = cube
COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o

all:$(TARGET)

$(TARGET):$(OBJS)
$(COM) $(CFLAGS) $(TARGET).cpp -o $^
$(OBJS):$(TARGET).cpp
$(COM) -c $<

clean:
        rm -f *.o
```

3.3.1 分割したファイルのコンパイル

ファイルを次の三つに分割する。

head.h

```
int cube(int) ;
```

sub.cpp

```
#include "head.h"
int cube(int a){
  return a*a*a ;
}
```

main.cpp

```
#include "head.h"
#include <iostream>
int cube(int a) ;

int main(int argc, char** argv){
  std::cout << cube(4) << std::endl ;
  return EXIT_SUCCESS ;
}
```

これら三つのファイルをつかったコンパイルの方法は下記の通りである。

```
$ ls
head.h main.cpp sub.cpp
$ c++ -c main.cpp
$ c++ -c sub.cpp
$ ls
head.h main.cpp main.o sub.cpp sub.o
$ c++ -o main main.o sub.o
$ ls
head.h main main.cpp main.o sub.cpp sub.o
```

また、これらの動作をひと綴りにした Makefile を Makefile2 に示す。

Makefile2

```
TARGET = main
TARGET2= sub
HEAD   = head

COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o $(TARGET2).o

all:$(TARGET)

$(TARGET):$(OBJS)
        $(COM) $(CFLAGS) -o $(TARGET) $(OBJS)

$(TARGET).o:$(TARGET).cpp $(HEAD).h
        $(COM) -c $<

$(TARGET2).o:$(TARGET2).cpp $(HEAD).h
        $(COM) -c $<
clean:
        rm -f *.o
```

Makefile2 の使い方は下記の通り、

```
$ make -f Makefile2
c++ -c main.cpp
c++ -c sub.cpp
c++ -Wall -o main main.o sub.o
$ ./main
64
```

インクルドガード

工事中。詳しくはググれ。

Makefile3

```
TARGET = main
TARGET2= sub
HEAD   = head

COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o $(TARGET2).o

all:$(TARGET)

$(TARGET):$(OBJS)
        $(COM) $(CFLAGS) -o $(TARGET) $(OBJS)
.cpp.o:
        $(CC) $(CFLAGS) -c $<
$(TARGET).o : $(HEAD).h  # main.cpp の依存ファイルは head.h
$(TARGET2).o: $(HEAD).h # main.cpp の依存ファイルは head.h

clean:
        rm -f *.o
```

Makefile3 の使い方は下記の通り、

```
$ make -f Makefile3
c++ -c main.cpp
c++ -c sub.cpp
c++ -Wall -o main main.o sub.o
$ ./main
64
```

3.4 Makefile その 3:引数を利用する

ファイルを実行する時に引数を与えたくなるだろう。その簡単な方法を示す。いままでおまじないとして main の引数に int 型の argc、char**型の argv を記入していた。

- argc
  実行ファイル + 与えた引数の数を表す。
  引数が 0 ならば argc=1、引数が 1 つならば argc=2、引数が 2 つならば argc=3、

- argv
  与えた引数を文字列として格納している。
  argv[0] は実行ファイル名、argv[1] は第 1 引数、argv[2] は第 2 引数といった具合である。

3.4.1 とりあえず動かす

引数の数と引数の文字列を表記するサンプルプログラム show.cpp と、その Makefile を次に示す。

show.cpp

```
#include <iostream>
int main(int argc,char** argv){
  std::cout << "argc = " << argc << std::endl ;
  for(int i=0; i< argc ;i++){
    std::cout << "argv[" << i << "]= " << argv[i] << std::endl;
  }
  return EXIT_SUCCESS ;
}
```

Makefile

```
TARGET = show

COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o

all:$(TARGET)

$(TARGET):$(OBJS)
$(COM) $(CFLAGS) -o $(TARGET) $(OBJS)
.cpp.o:
$(CC) $(CFLAGS) -c $<
$(TARGET).o :

clean:
rm -f *.o
```

実行例は次のとおりである。まずは make。

```
$ ls
Makefile show.cpp
$ make
cc -Wall -c show.cpp
c++ -Wall -o show show.o
```

次に引数を与えずに実行してみよう。

```
$ ./show
argc = 1
argv[0]= ./show
```

引数を与えていないので、argc は実行ファイルのみをしめす 1、argv[0] は今実行した実行ファイル名になっている。
次に引数を 3 つ与えてみる。

```
$ ./show a b c
argc = 4
argv[0]= ./show
argv[1]= a
argv[2]= b
argv[3]= c
```

verb—argc—は実行ファイル以外に 3 つの引数を持つことを示す 4、argv[i] はそれぞれ実行ファイル名を引数に対応する文字列である。
文字列の大きさは 1 文字でなくても良い。

```
$ ./show abc
argc = 2
argv[0]= ./show
argv[1]= abc
```

数字を引数に与えてみるとどうなるだろうか？

```
$ ./show a. 1.e-2
argc = 3
argv[0]= ./show
argv[1]= a.
argv[2]= 1.e-2
```

ここで注意して欲しいのは、argv[2] は文字列として出力されているということだ。例えば ROOT のコマンドライン上で 1.e-2 を入力すると、

```
root [] 1.e-2
(const double)1.00000000000000002e-02
```

と表示されるが、先ほどの例では入力した文字列のままで出力されていることに注意しておく必要が有る。したがって、argv[i] を数字として直接扱うことはバグの原因となりえるので注意すること。argv[i] を数字に変換する方法は練習問題とする。

3.4.2 練習

1. 実行ファイル自身以外の引数の数が 2 以外だとエラーメッセージを返すようにせよ。
2. 2 つの引数の和を表示するように変更せよ。ただし、入力される引数は数字だけが入力されると仮定してよい。(より応用な問題としては、そもそも引数が数字かどうかの判定なども考えられる。興味に応じて調べよ。)

   ヒント http://www.cplusplus.com/reference/cstdlib/atof/

3.4.3 解答例

1. 実行ファイル自身以外の引数の数が 2 以外だとエラーメッセージを返すようにせよ。
2. 2 つの引数の和を表示するように変更せよ。ただし、入力される引数は数字だけが入力されると仮定してよい。

showsol1.cpp

```
#include <iostream>
int main(int argc,char** argv){
  std::cout << "argc = " << argc << std::endl ;
  if(argc!=3){
    std::cerr << "Usage like ... \n ./showsol1 3.2 4" << std::endl ;
    return EXIT_FAILURE ;
  }
  for(int i=0; i< argc ;i++){
    std::cout << "argv[" << i << "]= " << argv[i] << std::endl;
  }
  std::cout << argv[1] << " + " << argv[2] << " = " << atof(argv[1]) + atof(argv[2]) << std::endl;
  return EXIT_SUCCESS ;
}
```

実行結果は下記の通りである。

```
$ ./showsol1
argc = 1
Usage like ...
 ./showsol1 3.2 4
$ ./showsol1 6 4
argc = 3
argv[0]= ./showsol1
argv[1]= 6
argv[2]= 4
6 + 4 = 10
```

3.5 Makefile その 4:ROOT のネイティブプログラミング

本章で ROOT のライブラリなどネイティブでも使用できるようにする。

3.5.1 ROOT ライブラリとのリンク

ROOT の数式ライブラリに含まれている定義値 π を表示しよう。

pi.cpp

```
#include "TMath.h"
#include <iostream>
int main(int argc, char** argv){
  std::cout << "Pi = " << TMath::Pi() << std::endl ;
  return EXIT_SUCCESS ;
}
```

メイクファイルを今までと同じように準備してみよう。

pi.cpp

```
TARGET = pi

COM    = c++
CFLAGS = -Wall
OBJS   = $(TARGET).o

all:$(TARGET)

$(TARGET):$(OBJS)
        $(COM) $(CFLAGS) -o $(TARGET) $(OBJS)
.cpp.o:
        $(CC) $(CFLAGS) -c $<
$(TARGET).o :

clean:
        rm -f *.o
```

make を実行してみると、

```
$ make
cc -Wall -c pi.cpp
pi.cpp:1:10: fatal error: 'TMath.h' file not found
#include "TMath.h"
         ^
1 error generated.
```

```
make: *** [pi.o] Error 1
```

はい、エラーが出ました。そこで Makefile を次のように書き換える。

Makefile2

```
TARGET = pi

COM        = c++
CFLAGS     = -Wall
ROOTCFLAGS = $(shell root-config --cflags)
ROOTLIBS   = $(shell root-config --libs)
CXXFLAGS   = $(ROOTCFLAGS) $(CFLAGS)
CXXLIBS    = $(ROOTLIBS)

OBJS   = $(TARGET).o

all:$(TARGET)

$(TARGET):$(OBJS)
$(COM) $(CXXLIBS) -o $(TARGET) $(OBJS)
.cpp.o:
$(CC) $(CXXFLAGS) -c $<
$(TARGET).o :

clean:
rm -f *.o
```

行った変更はコンパイルフラグを追加したことと、ROOT のライブラリとのリンク作業である。make してみよう。

```
$ make -f Makefile2
cc -pthread -m64 -I/usr/local/hep/root/v5.34.09/include/root -Wall -c pi.cpp
c++ -L/usr/local/hep/root/v5.34.09/lib/root -lCore -lCint -lRIO -lNet -lHist \
-lGraf -lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lMathCore \
-lThread -lpthread -lm -ldl -o pi pi.o
$ ./pi
Pi = 3.14159
```

ちょっと長ったらしい実行分の後、目的の実行ファイル pi が出来上がる。

3.5.2 幾つかのサンプルプログラム

ヒストグラムの描写と保存

hist2.cpp

```
#include "TCanvas.h"
#include "TH1.h"
#include <iostream>

int main(int argc, char** argv){
  TCanvas *c1 = new TCanvas("c1", "c1", 600, 600) ;
  TH1D *h = new TH1D("h", "h", 100, -5., 5.) ;
  h->FillRandom("gaus") ;
  h->Draw("E") ;
  c1->SaveAs("hist2c1.eps") ;
  delete h ;
  delete c1 ;
  return EXIT_SUCCESS ;
}
```

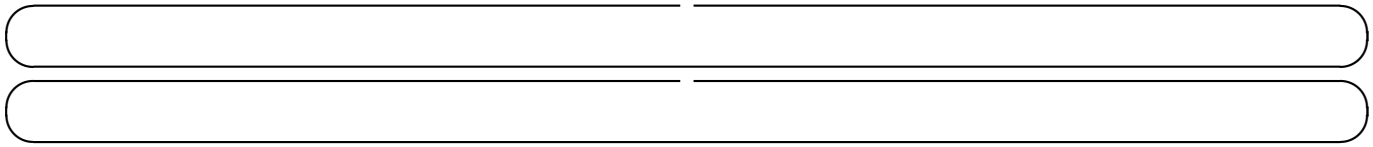
Makefile

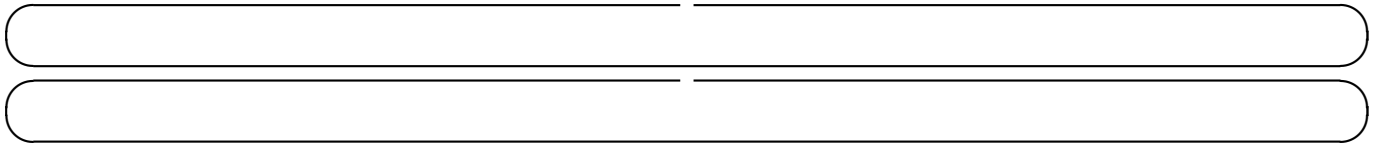
```
TARGET = hist2

COM        = c++
CFLAGS     = -Wall
ROOTCFLAGS = $(shell root-config --cflags)
ROOTLIBS   = $(shell root-config --libs)
CXXFLAGS   = $(ROOTCFLAGS) $(CFLAGS)
CXXLIBS    = $(ROOTLIBS)

OBJS   = $(TARGET).o

all:$(TARGET)

$(TARGET):$(OBJS)
        $(COM) $(CXXLIBS) -o $(TARGET) $(OBJS)
.cpp.o:
        $(CC) $(CXXFLAGS) -c $<
$(TARGET).o :

clean:
        rm -f *.o
```

実行方法は下記の通り。

```
$ ls
Makefile hist2.cpp
$ make
cc -pthread -m64 -I/usr/local/hep/root/v5.34.09/include/root -Wall -c hist2.cpp
c++ -L/usr/local/hep/root/v5.34.09/lib/root -lCore -lCint -lRIO -lNet -lHist -lGraf \
-lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics -lMathCore -lThread -lpthread \
 -lm -ldl -o hist2 hist2.o
$ ./hist2
Info in <TCanvas::Print>: eps file hist2c1.eps has been created
$ ls
```

```
Makefile hist2c1.eps hist2 hist2.cpp hist2.o
```





Appendix A

補足

A.1 名前空間

「あめ」といった時にそれがどういう内容を表すだろうか？識別する方法としては、

1. 「「気象現象」に属する「あめ」」
2. 「「食べ物」に属する「あめ」」

などとしてしまえば、「あめ」の表す内容は明確になる。この時の「気象現象」や「食べ物」のようなくくりに当たる概念が名前空間である。この時使用した「属する」という言葉を C++ではスコープ演算子"::"で表す。つまり、先程の例を C++風に表現すると下記のようになる。

1. 気象現象::あめ
2. 食べ物::あめ

A.1.1 名前空間 std::

プログラム中で

```
#include <iostream>
```

と宣言すれば、名前空間 std が使用可能となる。具体的な使い方としては、

```
std::cout << "abc" << std::endl;
```

などである。意味としては、abc という文字列と std::endl という改行命令を std::cout で設定されている標準出力外面へ出力する。

A.1.2 名前空間 TMath::

プログラム中で

```
#include "TMath.h"
```

と宣言すれば、名前空間 TMath が使用可能となる。http://root.cern.ch/root/html/TMath.html またコマンドライン上で ROOT を使用する時には宣言の必要はない。

```
root [] TMath::C()
(Double_t)2.99792458000000000e+08
root [] TMath::Pi()
(Double_t)3.14159265358979312e+00
root [] TMath::Power(2,3)
(Double_t)8.00000000000000000e+00
```

```
root [] TMath::Abs(-2.)
(Double_t)2.00000000000000000e+00
root [] TMath::Sin(1.)
(Double_t)8.41470984807896505e-01
```

などである。上記の入力や引数や返り値の意味することは各自で調べよ。

A.2 ROOT で使う色

http://root.cern.ch/root/html/TAttFill.html#F1

Figure A.1: ROOT で使用できる色

A.3 ROOT で使うスタイル

http://root.cern.ch/root/html/TAttFill.html#F2

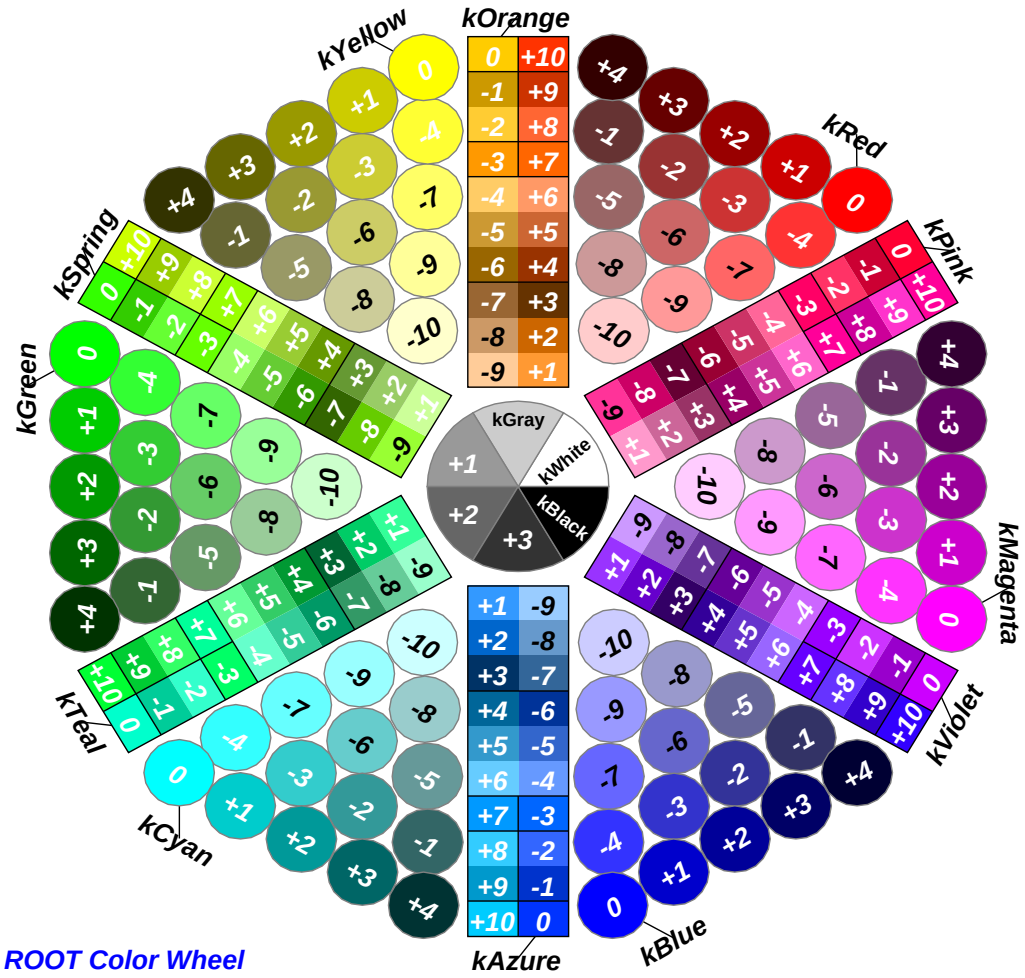


Figure A.2: ROOT で使用できる色

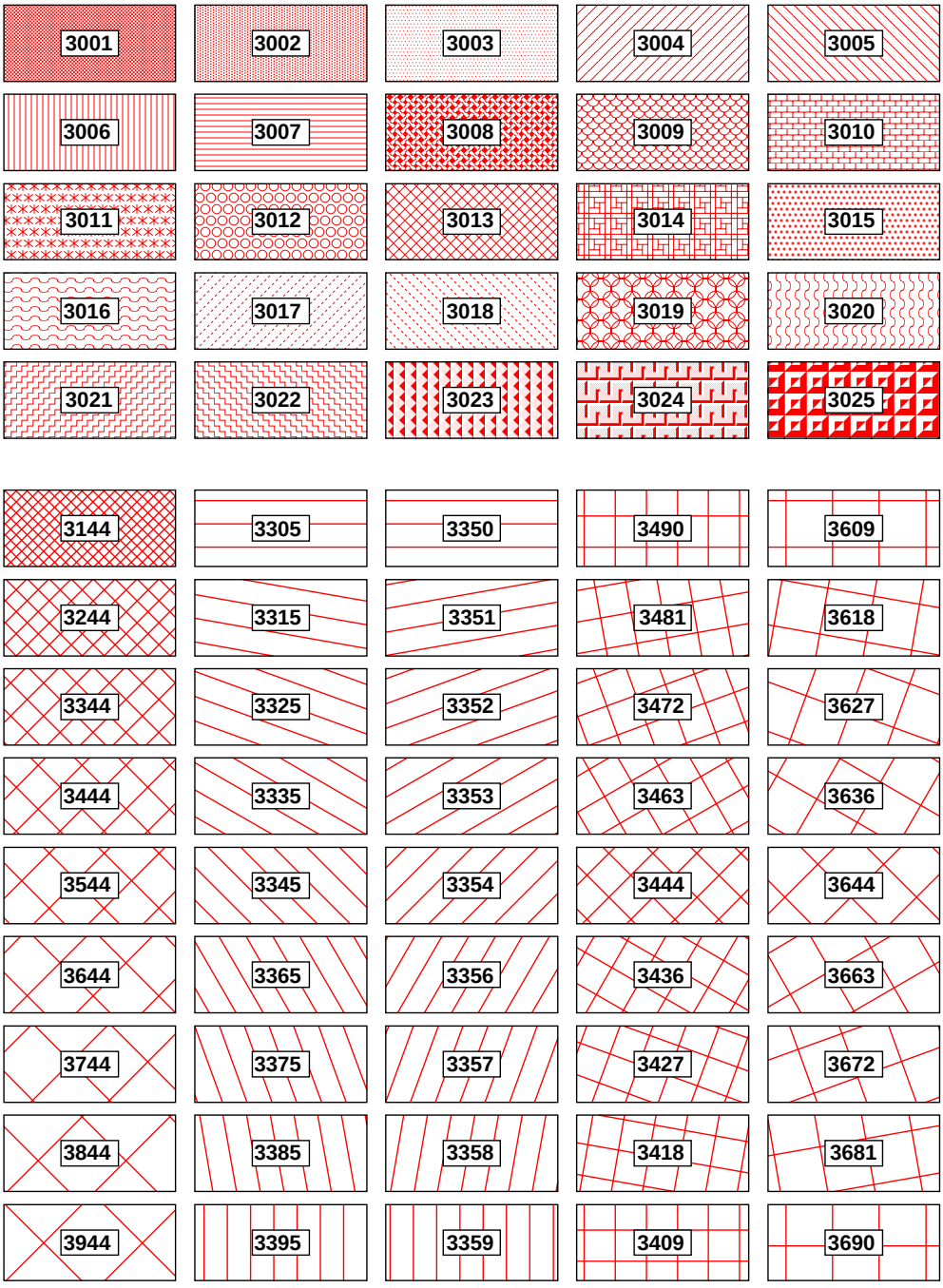


Figure A.3: ROOT で使用できるスタイル