粒子物理与核物理实验中的数据分析

王喆

清华大学

第二讲： Linux，C++

本讲摘要

- 面对对象的程序设计
- C++的类，实例，指针
- 类的继承，重用
- 朋友类
- 类的多态
- 编译并执行C++程序
- Makefile简介

后继课程的准备要求

- 有能够使用的Linux，自行安装或可以登录其他服务器
- 可以编译链接使用C++
- 安装ROOT
- 学会安装一些支撑软件（各个系统不一样）
- 下半学期稍后我们要安装Geant4

大家可以提前着手准备

本节课的例子

登录到一个Linux

- Windows to Linux

使用ssh客户端程序(XManager,SecureCRT, putty...)

- Linux to Linux
- Virtual Box

文件解压缩

- tar cvfz Lec2.tgz Lec2　　压缩
- tar xvfz Lec2.tgz　　解压

本节的例子在Lec2 Attachment 下载后需要解压

安装新的工具包(Ubuntu, CentOS不一样)

- sudo apt-get install g++
- sudo apt-get install emacs23 (或vi)

C++ 的历史

- C 语言大约在 1970 年诞生于Bell Labs
  UNIX 的一部分是用C语言写的
- Bjarne Stroustrup 在80年代基于 C语言开发了 C++
  “C with classes”, 也就是说允许面向对象编程的用户自定义的数据类型”

- C++ 是作为C的增量改进版而开发的
- C 可以看做 C++ 的子集，所以，大部分C程序可以直接用C++编译器编译
- 从开发以来有4个重要 C++ 标准： C++98 (1998), C++03 (2003) and C++11 (2011) and C++14 (2014).

设计带给我们变革

- 例子：
  - 小学时我们要解应用题，思路要清晰，好多加减乘除相当复杂
    - 难倒成年人
  - 初中时我们学会了列方程，这样想法就简单多了
    - 备注：问题是我们要会解
  - 后来我们又学了微积分，思路就更开阔了
    - 备注：我们还是要能够求解

程序语言设计

- 计算机编程语言：
  - 汇编语言：生涩难懂，想要开发高级或复杂的功能等于不可能
- 有了C语言，终于轻松的写a+b了
  - C语言编译器会帮我们把a+b写成机器语言。而且编译器可以深度优化，是实现效率大幅提高
  - 擅长写函数

汇编：

```
RAND   PROC
       PUSH AX
       STI
       MOV AH,0
```

C语言：

```
c=a+b;
```

面对对象的程序设计

- C++
  - 对象：英语Object
    应该翻译成“客体或物体”
    面对客体的程序设计，
    基于物理特征的编程。
  - 直接描述一个物体，描述它的属性，功能。
  - 思路更直接，顺畅
    （不用再解应用题）
  - C++编译器，可以翻译，解释我们的逻辑给电脑

面对对象设计
C++

狗
- 吠叫:
- 毛皮颜色
- 品种

粒子
- 衰变
- 类型
- 寿命
- 质量

C++速成，例子**1.** HelloWorld

C++速成：HelloWorld

首先用emacs/vi，编写包含以下内容的文件 HelloWorld.cc

```
// A C++ program
#include <iostream>
using namespace std;
int main(){
  cout << "Hello World!" << endl;
  return 0;
}
```

```
emacs -nw HelloWorld.cc
   editing …
save and exit:
   ctrl+x, ctrl+c
save:
   ctrl+x, ctrl+w
```

然后对文件进行编译形成机器可读的代码：

```
> g++ -o HelloWorld HelloWorld.cc
```

调用编译器 (gcc)　　输出的文件名　　源代码

最后执行程序

```
>./HelloWorld
Hello World!
```

← 用户键入(注意：>为系统提示符)
← 计算机显示结果

C++速成，例子2. VolCuboid

一个较好的C++程序组织结构

Lec2/VolCuboid/

```
[training ~/DataAnalysis/Lec2/VolCuboid]$ ls
bin/        compile.sh*  Makefile       Makefile.not.easy  src/
build.sh*   include/     Makefile.easy  obj/
```

Linux下标准的**C++**程序项目一般把源文件、头文件、目标文件及可执行文件放在不同目录，便于维护管理。

比如某个程序项目，为该项目建立工作目录(如VolCuboid)，工作目录中一般会有bin, include, obj, src等子目录，分别存放可执行文件、头文件、目标文件和源文件。工作目录中还会有编译文件以及其它辅助文件(如输入参数文件)。

C++类（定义，头文件）

Lec2/VolCuboid/include/VolCuboid.h

```
#ifndef VOLCUBOID_H
#define VOLCUBOID_H

#include <iostream>
//#include <math>

class VolCuboid {
    public:
        VolCuboid(float x, float y, float z);
        ~VolCuboid();//Deconstructor function
        float Vol();//Member Function
        float Area();//Member Function
    private:
        float length, width, height;
};

#endif
```

构造　　析够

成员变量　　面积　　体积

C++类（实现，执行，源文件）

Lec2/VolCuboid/src/VolCuboid.cc

```
#include "VolCuboid.h"

VolCuboid::VolCuboid(float x, float y, float z) {
    length = x ;
    width  = y ;
    height = z ;
}

VolCuboid::~VolCuboid() {
    //new pointers should be deleted here.
    //if not, do nothing.
}

float VolCuboid::Vol() {
    return length*width*height;
}

float VolCuboid::Area() {
    float area;
    area = 2*length*width + 2*length*height + 2* width*height ;
    return area;
}
```

构造函数功能实现了

真正实现了体积，面积的计算

C++类（使用，主函数）

Lec2/VolCuboid/src/main.cc

```
//#########main.cc###################

#include <iostream>
//#include <math>

#include "VolCuboid.h"
//#include "TH1F.h"

using namespace std;

int main ()
{
   cout << "Class VolCuboid "<< endl;
   float length, width, height;

   length = 2.0 ; //cm
   width  = 3.0 ; //cm
   height = 4.0 ; //cm

   VolCuboid myVolCuboid( length, width, height );
   //VolCuboid *myVolCuboid = new VolCuboid( length, width, height );

   float volume = myVolCuboid.Vol() ;
   //float volume = myVolCuboid->Vol() ;
   cout << "Volume is " << volume << " cm^3" << endl;
   cout << "Area   is " << myVolCuboid.Area() << " cm^2" << endl;
```

包含头文件，使主函数可以找到类的定义

生成类的实例，使用方法

C++类（指针）

Lec2/VolCuboid/src/main.cc

```
// use pointer
VolCuboid * pVolCuboid = new VolCuboid( length, width, height );

volume = pVolCuboid->Vol();
cout << endl;
cout << "Operation with pointer" << endl;
cout << "Volume is " << volume << " cm^3" << endl;
cout << "Area   is " << pVolCuboid->Area() << " cm^2" << endl;
```

也可以生成指针引用

指针里面放的是一个数据在计算机内存中的地址。例如大家知道我的办公室地址804就可以找到我了。

计算机的内存分成多个段（segment），数据存储的，程序代码的，每个程序还有自己的段内存空间，不能跑到别人那去，去了就叫段错误，segment fault，根本原因是某个指针被赋予了一个非法的值。

类的定义和使用的重要关键点

0. 关键字 class
1. 构造函数，
2. 析构函数
3. 成员变量
4. 成员函数
5. 实现成员函数的功能
6. 类要生成实例才能使用

使用刚才的这个例子：

1. 利用g++来编译，而且写成了脚本

例如 Lec2/VolCuboid/build.sh 中的内容

```
#!/bin/tcsh

g++ -o bin/try -Iinclude/ src/*.cc
```

2. 这样脚本还不够智能和快捷，而且功能太差，我们要使用Makefile

Lec2/VolCuboid/Makefile
try：
> make
> bin/VolCub （或者 ./bin/VolCub)

Makefile简介

Lec2/VolCuboid/Makefile

语法很复杂，但需要改动的地方很少

```
# # setup control #
TOP := $(shell pwd)/
OBJ := $(TOP)obj/
BIN := $(TOP)bin/
SRC := $(TOP)src/
INCLUDE := $(TOP)include/
#CPPLIBS =
#INCLUDE+=

# # set up compilers #
CPP = g++
CPPFLAGS = -O -Wall -fPIC -I$(INCLUDE)

######## Make Executables #####
all: VolCub
VolCub : $(patsubst $(SRC)%.cc,$(OBJ)%.o,$(wildcard $(SRC)*.cc))
        $(CPP) $^ $(CPPLIBS) -o $(BIN)$(notdir $@)
@echo
#########################
$(OBJ)%.o : $(SRC)%.cc
        $(CPP) $(CPPFLAGS) -c $(SRC)$(notdir $<) -o $(OBJ)$(notdir $@)
@echo
.PHONY:clean
        clean: rm -f $(OBJ)*.o rm -f $(BIN)*
```

头文件或者库文件目录

g++命令的参数

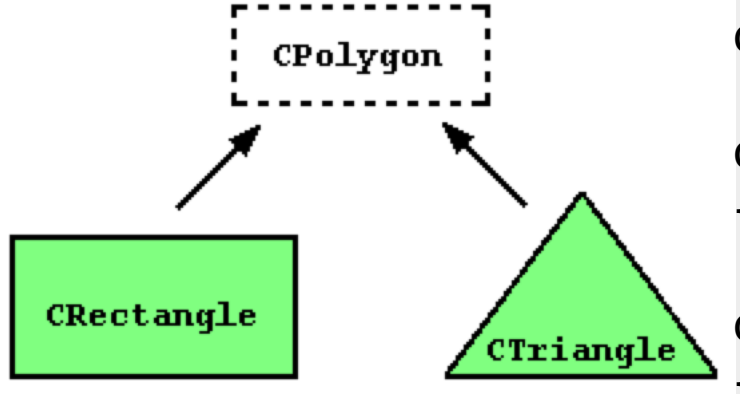
可执行文件

C++后缀,如所有.cc改为.o

类的重要概念和应用

继承 Inheritance

- ***Inheritance*** 是面向对象编程最重要的特性之一
- 类可以被扩展，即可以创建一个类使其保持“基类”的所有属性 ➔ ***inheritance***
- 关于 ***“基类” base class*** 和 ***“派生类” derived class***: 派生类继承基类的成员，以此为基础还可以添加新的成员。




```
class CPolygon {/*…*/};

class CRectangle: public CPolygon
{/*……*/};

class CTriangle: public CPolygon
{/*……*/};
```

```
class derived_class_name: public base_class_name
{ /*...*/ };
```

继承示例

```
// derived classes
#include <iostream>
using namespace std;

class Polygon {    //base class
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b;}
};
class Rectangle: public Polygon {
  public:                    //derived class
    int area ()
      { return width * height; }
};
class Triangle: public Polygon {    //derived class
  public:    int area ()
      { return width * height / 2; }
};
```

```
int main () {
  Rectangle rect;
  Triangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

- *Polygon* 是基类
- *Rectangle* 和 *Triangle* 是基类 *Polygon* 的派生类
- *Rectangle* 和 *Triangle* 可直接使用 *Polygon* 的public和protected成员

- **注：派生类不可访问基类的 private 成员**

继承的要点

基类的哪些属性被派生类继承了？

- 原则上，派生类会继承基类的所有成员，除了基类的:
  - constructors 和 destructor
  - assignment operator members (operator=)
  - friends
  - private members

多重继承

- 派生类可以继承自多个基类，不同基类之间用逗号分隔。

```
class Rectangle: public Polygon, public Output;
class Triangle: public Polygon, public Output;
```

一些基本概念的测试

下面这些表达式是什么意思？

Statement:
`int A::b(int c) { }`
`a->b`
`class A: public B {};`

友元 Friendships

- In principle, ***private*** and ***protected*** members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not apply to "friends".
- Friends are functions or classes declared with the ***friend*** keyword.
- A non-member function can access the private and protected members of a class if it is declared a friend of that class. That is done by including a declaration of this external function within the class, and preceding it with the keyword friend

友元函数示例

```
// friend functions
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle() {}
    Rectangle (int x, int y) : width(x), height(y) {}
    int area() {return width * height;}
    friend Rectangle duplicate (const Rectangle&);
};

Rectangle duplicate (const Rectangle& param){
  Rectangle res;
  res.width = param.width*2;
  res.height = param.height*2;
  return res;
}
```

```
int main () {
  Rectangle foo;
  Rectangle bar (2,3);
  foo = duplicate (bar);
  cout << foo.area() << '\n';
  return 0;
}
```

- *duplicate* is a friend function of class Rectangle
- It returns an object of class *Rectangle*
- It can access private data member of class *Rectangle* (*width, height*)

友元函数示例

```
// friend class
#include <iostream>
using namespace std;
class Square;

class Rectangle {
    int width, height;
  public:
    int area ()
      {return (width * height);}
    void convert (Square a);
};

class Square {
  friend class Rectangle;
  private:
    int side;
  public:
    Square (int a) : side(a) {}
};
```

← what?

```
void Rectangle::convert (Square a) {
  width = a.side;
  height = a.side;
}

int main () {
  Rectangle rect;
  Square sqr (4);
  rect.convert(sqr);
  cout << rect.area();
  return 0;
}
```

- *Rectangle* is a friend class of class *Square*, and can access class *Square*'s data member(*sides*)
- Notice:
  1) Direction of friendship
  2) friendship NOT transitive

Polymorphism (多态性)

指向基类的指针

- One of the key features of class inheritance is that a pointer to a derived class is type-compatible with a pointer to its base class.
- Polymorphism is the art of taking advantage of this simple but powerful and versatile feature.

```
base class:     Mother
derived class: Daughter

Daughter myD;
Mother *myM = &myD;
```

指向基类的指针

```
// pointers to base class
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
};
class Rectangle: public Polygon {
  public:
    int area()
      { return width*height; }
};
class Triangle: public Polygon {
  public:
    int area()
      { return width*height/2; }
};
```

```
int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

- **ppoly1** and **ppoly2** are pointers of **Polygon** class
- They are assigned the addresses of **rect** and **trgl**, objects of type **Rectangle** and **Triangle**
- **They can only access *inherited* members!**

ppoly1->area(); ✗

如何访问函数area()

- If **area()** is defined in the base class **Polygon**…
- But the implementations of **area()** in **Rectangle** and **Triangle** are different
- **Virtual member** as a solution:
  - A member function that can be redefined in a derived class, while preserving its calling properties through references
  - The syntax for a function to become virtual is to precede its declaration with the **virtual** keyword

虚成员 virtual members 示例

```
// virtual members
#include <iostream>
using namespace std;

class Polygon {
  protected:
    int width, h
  public:
    void set_val
      { width=a;
    virtual int area ()
      { return 0; }
};

class Rectangle: public Polygon {
  public:
    int area ()
      { return width * height; }
};
```

```
class Triangle: public Polygon {
  public:
    int area () { return (width * height / 2); }
};
int main () {
  Polygon * ppoly3 = &poly;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly3->set_values (4,5);
  cout << ppoly1->area() << '\n';
  cout << ppoly2->area() << '\n';
  cout << ppoly3->area() << '\n';
  return 0;
}
```

想想我们模拟粒子时候是不是很方便！

question: what are the output?

More words about virtual members

- Non-virtual members can also be redefined in derived classes
- But non-virtual members of derived classes cannot be accessed through a reference of the base class

- A class that declares or inherits a virtual function is called a **polymorphic class**

抽象基类

- **Abstract base classes** are classes that can only be used as base classes
- They are allowed to have virtual member functions without definition (known as **pure virtual functions**)
- The syntax is to replace their definition by =0

```
// abstract class CPolygon
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area () =0;
};
```

- Abstract base classes cannot be used to instantiate objects

  Polygon mypolygon; 

  Polygon *ppolygon; 

模板 Template

- In C++, two different functions can have the same name if their parameters are different
  - either because they have a different number of parameters
  - or because any of their parameters are of a different type
- They are called overloaded functions
- Template of functions is convenient

```
// function template
#include <iostream>
using namespace std;
template <class T>
T sum (T a, T b){
  T result;
  result = a + b;
  return result;
}
int main () {
  int i=5, j=6, k;
  double f=2.0, g=0.5, h;
  k=sum<int>(i,j);
  h=sum<double>(f,g);
  cout << k << '\n';
  cout << h << '\n';
  return 0;
}
```

ROOT安装

1. 下载root的源代码http://root.cern.ch/

找到Download，还有Documentation->Building root

2. 如上提示的方法解压，安装

3. 你的系统可能会缺少一些支持软件

http://root.cern.ch/drupal/content/build-prerequisites

4. 在Ubuntu环境中可以利用apt-get命令安装缺少的内容。一般缺少的都是开发包，例如libglew1.5-dev，即头文件，库文件，链接库等。

演示ROOT

- > root

我们看到类在这里有着充分的应用！

演示1：

- root [0] TH1F h1("h1","myhistorgram", 100, -5., 5.)
- root [1] h1.FillRandom("gaus",5000)
- root [2] h1.Draw()

演示2：

- root [3] TF1 poi("poi","5**int(x)*exp(-5)/ TMath::Factorial(int(x))",0,20)
- root [4] poi.Draw()

演示3：

- 其他

小结

- C++
- 类，多态
- g++编译C++程序
- 用Makefile编译C++程序
- ROOT功能初试

作业

- 对刚才的类添加新的功能
  计算它可以容纳的最大的球的体积，输出。
- 理解课上的类的继承的重要的概念，编译链接通过，并成功运行课上26，27，29，31，33页的例子。
- 脑力锻炼题目：利用C或者C++编程，利用递归方式解决20层的Hanoi Tower问题，打印输出答案。
- 为什么我们需要多态？列举几种应用场景。简要研读C++发展简史。