# 粒子物理与核物理实验中的数据分析

王　喆　　杨振伟
清华大学

第4讲：C++

# Outline

- 重新讨论类
- operator
- make & Makefile
- gdb
- STL

# 例子：Lec2/VolCuboid/

文件及目录结构：

```
[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**

1. 源文件
2. 头文件
3. 目标文件
4. 可执行文件

**放在不同目录，便于维护管理。**

# 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

成员函数

成员变量

# 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，并使用

# 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。

为什么要使用指针：一个object在内存里面的复制操作太消耗资源了。传递一个单值的指针则很方便

# 常量，常指针

```
const int pi = 3.1415926;
const VolCuboid & ref = aVolCuboid;
```

变量pi, ref不得更改

```
VolCuboid::Area() const
```

Area函数不得修改类VolCuboid的成员变量

```
const VolCuboid* pVol;
```

指针所指的内容是常量

```
VolCuboid* const pVol;
```

指针本身是常量

```
const VolCuboid* const pVol;
```

指针的值和指针所指的内容全部是常量

这样做的目的：设计意图明确，可读性更强，减少出错概率

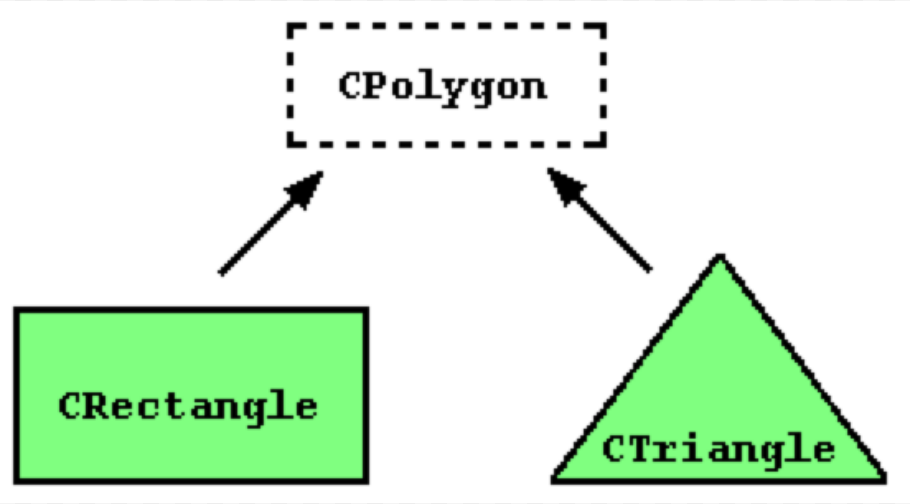
# 一些基本概念的测试

下面这些表达式是什么意思？
名字空间？

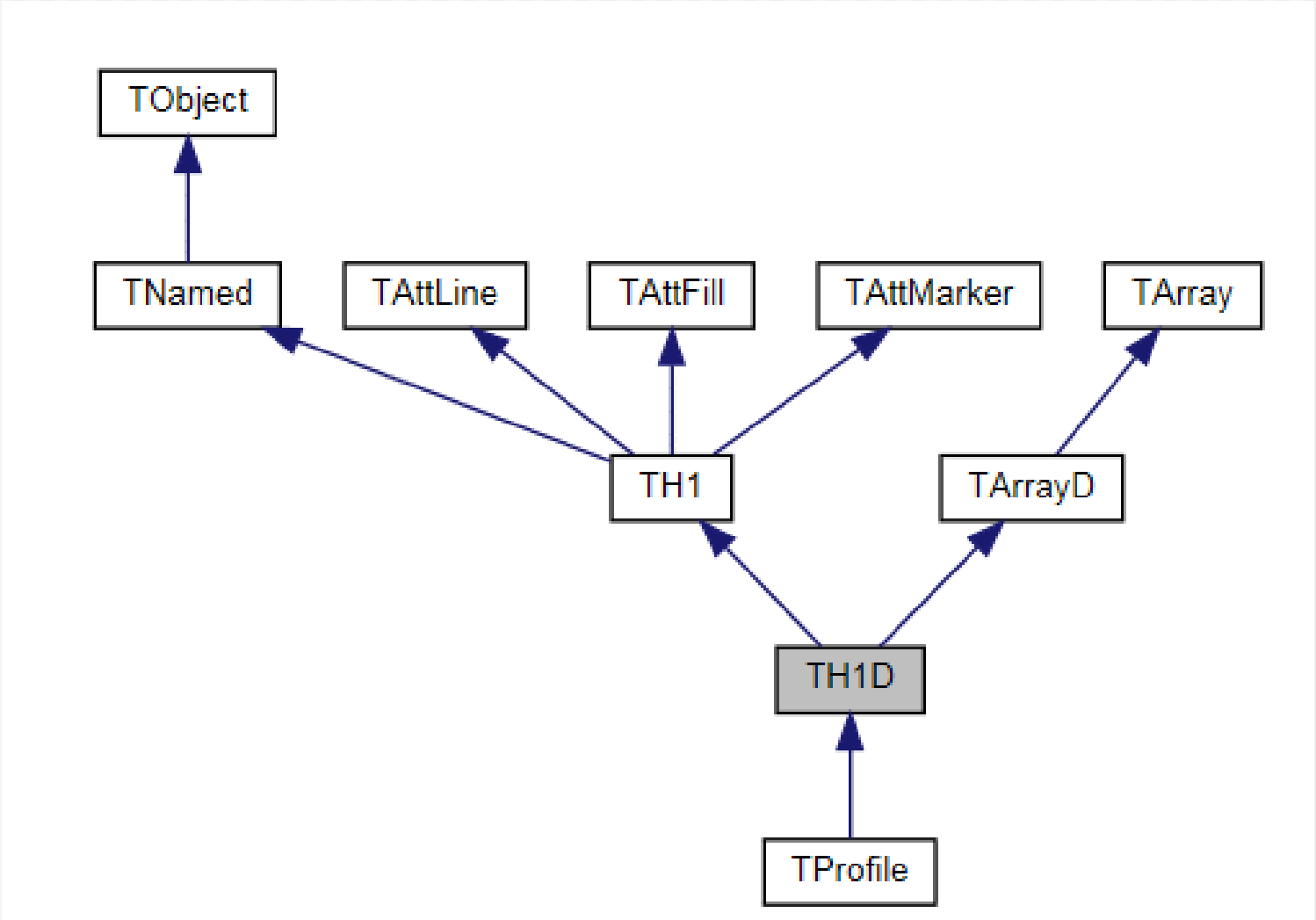
| Statement: |
| --- |
| `int A::b(int c) { }` |
| `a->b` |
| `class A: public B {};` |

# 类，继承，虚函数，多态

类：让我们更方便的描述一些物体，对象，问题
类继承：方便的分层次，模块化，设计意图明确



# 一个root的关于继承的例子



# 虚成员 virtual members 示例

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

class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    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 () {
  Rectangle rect;
  Triangle trgl;
  Polygon poly;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  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;
}
```

question: what are the output?

# 虚基类，虚函数：Geant4的基本实现原理

模拟函数（一个形体，一个粒子，位移）
{
    一个形体的辐射长度？
    一个粒子的电荷？
    一个粒子在一个形体内的位移？
}

程序基础设计者可以完成如上的框架型代码，
其中**一个形体，一个粒子，全部是基类指针**

使用该函数：
模拟函数（正方形铁块，电子，位移）

**实际使用时：一个形体，一个粒子被赋予派生类指针**

虚基类和虚函数很好的完成这一设想

# 编译，链接

在Lec2/VolCuboid/中展示了多种编译链接的方式

build.sh:

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

compile.sh:

```
#!/bin/bash
#### compile cpp programs

g++ -c -I./include/ src/*.cc
g++ -o bin/try *.o
rm -f *.o
```

# 一个简单的Makefile

Makefile.easy

```
default: hello

hello:
        g++ -o bin/hello -Iinclude/ src/*.cc
clean:
        rm -f obj/*.o bin/*
```

在make命令后可以选择哪一个Makefile

> make -f Makefile.easy

还能选择哪个make目标

> make clean -f Makefile.easy

> make hello -f Makefile.easy

# 一个复杂一些Makefile

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

# A makefile with external libraries

```
# An example of makefile
TOP      := $(shell pwd)/
OBJ      := $(TOP)obj/
BIN      := $(TOP)bin/
SRC      := $(TOP)src/
INCLUDE   := $(TOP)include/

CPP      = g++
LD      = $(CPP)
CPPFLAGS = -O -Wall -fPIC -I$(INCLUDE)

ROOTCFLAGS    = $(shell root-config --cflags)
ROOTLIBS      = $(shell root-config --libs)
ROOTGLIBS     = $(shell root-config --glibs)
CPPFLAGS += -I$(ROOTCFLAGS)
CPPLIBS = $(ROOTLIBS) $(ROOTGLIBS)

######### Make Executables ############################
all: main
main    : $(patsubst $(SRC)%.cc,$(OBJ)%.o,$(wildcard $(SRC)*.cc))
        $(LD) $^ $(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)*
```

What if external libraries like ROOT is used?

What you need to do is to let the compiler know:
1) where is the head file?
2) where is the library file

You may do this manually or by the “root-config” command

**Try in the command line:**
**root-config --cflags**
**root-config --libs**
**root-config --glibs**

# Operators in class

Operator overload in class is useful.
E.g., you have a class “MyComplex” for complex numbers, and use it to instantiate two complex numbers:

MyComplex c1(1.0,2.0), c2(2.0,4.0);

You may want to assign the plus of them to “sum”:

MyComplex sum=c1+c2;

It would be convenient if you overload “+”

```
#ifndef MYCOMPLEX_H
#define MYCOMPLEX_H

class MyComplex {
    public:
        double real;
        double imag;
        MyComplex(double real, double imag);
        ~MyComplex();
        double Mod();
};

#endif
```

```
#include "MyComplex.h"
#include <cmath>

MyComplex::MyComplex(double re, double im) {
    real=re;
    imag=im;
}

MyComplex::~MyComplex(){;}

double MyComplex::Mod(){
    return sqrt(real*real+imag*imag);
}
```

# Operators in class

Operator overload in class is useful.
E.g., you have a class “MyComplex” for complex numbers, and use it to instantiate two complex numbers:

MyComplex c1(1.0,2.0), c2(2.0,4.0);

You may want to assign the plus of them to “sum”:

MyComplex sum=c1+c2;

It would be convenient if you overload “+”

```
#ifndef MYCOMPLEX_H
#define MYCOMPLEX_H

class MyComplex {
    public:
        double real;
        double imag;
        MyComplex(double real, double imag);
        ~MyComplex();
        double Mod();
};

#endif
```

# Overload of operator +

```
#ifndef MYCOMPLEX_H
#define MYCOMPLEX_H
#include <cmath>
class Complex {
    public:
        double real;
        double imag;
        Complex(double real, double imag);
        ~Complex();
        double Mod();
};

Complex::Complex(double re, double im) {
    real=re;
    imag=im;
}

Complex::~Complex(){;}

double Complex::Mod(){
    return sqrt(real*real+imag*imag);
}

Complex operator+(Complex c1,Complex c2) {
    return Complex(c1.real+c2.real,c1.imag+c2.imag);
}
#endif
```

```
#include <iostream>
#include <Complex.h>

using namespace std;

int main(){

    Complex c1(1.0,2.0);
    Complex c2(2.0,4.0);
    Complex sum=c1+c2;
    cout << "sum.Mod() = " << sum.Mod() << endl;

    return 0;
}
```

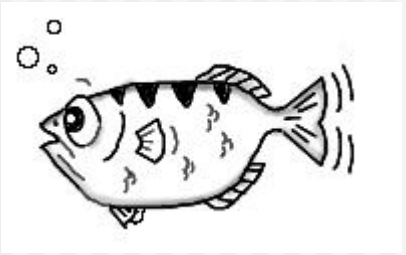
注意，注意：

C++已经有复数类了

#include <complex>

# GDB: the GNU project debugger

http://www.gnu.org/software/gdb/



GDB allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

What GDB can do:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

A nice quick-start example:

http://www.cnblogs.com/davidwang456/p/3450532.html

# GDB example

- You need compile the program with **“-g”** option

  **g++ main.cc -g -Wall -o main**

  **(-g: produce debug information)**

  **(-Wall: turns on all optional warnings)**
- Start your program with gdb:

  **gdb main**

  **gdb main pid**
- Try the following gdb command to see what happens

(gdb) break 11
(gdb) run
(gdb) step
(gdb) list
(gdb) watch n
(gdb) watch result
(gdb) continue
(gdb) backtrace
(gdb) frame numbe

(gdb) b 8 if i==10
(gdb) next
(gdb) info breaks
(gdb) disable
(gdb)
(gdb) print val
(gdb) next
(gdb) continue
(gdb) quit

# STL (Standard Template Library)

- A software library for the C++ programming language that influenced many parts of the C++ Standard Library
- It provides four components
  - **algorithms**
    - ✓ **Non-modifying sequence operations**
    - ✓ **Modifying sequence operations**
    - ✓ **Sorting**
    - ✓ **Merge**
    - ✓ **……**
  - **containers**
    - ✓ **array, deque, forward_list, list, map, queue, set stack, unordered_map, unordered_set, vector**
  - **functional**
  - **iterators**

http://www.cplusplus.com/reference/

# STL vector

为了可以使用vector，必须在你的头文件中包含下面的代码：

```
#include <vector>
```

vector属于std命名域的，因此需要通过命名限定，如下完成你的代码：

```
using std::vector;
vector<int> c;
```

或者连在一起，使用全名：

```
std::vector<int> c;
```

c.max_size()
返回容器中数据的数量。
c.pop_back()
删除最后一个数据。
c.push_back(elem)
在尾部加入一个数据。

vector和数组效率是差不多的，vector是可变长的，其他的一些操作更方便。

# vector的循环，迭代子

std::vector<*int*>::iterator
或者
std::vector<*int*>::const_iterator

```
// vector::begin/end
#include <iostream>
#include <vector>

int main ()
{
  std::vector<int> myvector;
  for (int i=1; i<=5; i++) myvector.push_back(i);

  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

# STL vector更方便，例如，排序，sort

```
// sort algorithm example
#include <iostream>     // std::cout
#include <algorithm>    // std::sort
#include <vector>       // std::vector

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  std::vector<int> myvector (myints, myints+8);               // 32 71 12 45 26 80 53 33

  // using default comparison (operator <):
  std::sort (myvector.begin(), myvector.begin()+4);           //(12 32 45 71)26 80 53 33

  // using function as comp
  std::sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)

  // using object as comp
  std::sort (myvector.begin(), myvector.end(), myobject);     //(12 26 32 33 45 53 71 80)

  // print out content:
  std::cout << "myvector contains:";
  for (std::vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

# STL map

```
// constructing maps
#include <iostream>
#include <map>

bool fncomp (char lhs, char rhs) {return lhs<rhs;}

struct classcomp {
  bool operator() (const char& lhs, const char& rhs) const
  {return lhs<rhs;}
};

int main ()
{
  std::map<char,int> first;

  first['a']=10;
  first['b']=30;
  first['c']=50;
  first['d']=70;

  std::map<char,int> second (first.begin(),first.end());

  std::map<char,int> third (second);

  std::map<char,int,classcomp> fourth;                 // class as Compare

  bool(*fn_pt)(char,char) = fncomp;
  std::map<char,int,bool(*)(char,char)> fifth (fn_pt); // function pointer as Compare

  return 0;
}
```

形成一组key和object对的列表，适应更多的应用。

# map的一系列功能

find，insert，operator[]，rbegin，rend

```cpp
// accessing mapped values
#include <iostream>
#include <map>
#include <string>

int main ()
{
  std::map<char,std::string> mymap;

  mymap['a']="an element";
  mymap['b']="another element";
  mymap['c']=mymap['b'];

  std::cout << "mymap['a'] is " << mymap['a'] << '\n';
  std::cout << "mymap['b'] is " << mymap['b'] << '\n';
  std::cout << "mymap['c'] is " << mymap['c'] << '\n';
  std::cout << "mymap['d'] is " << mymap['d'] << '\n';

  std::cout << "mymap now contains " << mymap.size() << " elements.\n";

  return 0;
}
```

- 存在，就赋值
- 不存在，生成，再赋值

# 作业

1. 编辑，并运行第26页关于vector的程序
   打印程序，及运行结果

2. 对第12页的程序，利用-g编译选线，用dgb调试，
   验证确实调用了派生类的成员函数Area()