

# 粒子物理与核物理实验中的数据 分析

---

杨振伟

清华大学

第一讲：Linux环境下编程(2)

# 上讲回顾(1)

---

## ■ Linux简介

发行版, Scientific Linux CERN(SLC),  
强大的命令行模式

## ■ 登录Linux服务器

从Linux登录:

```
ssh [-X] yangzw@training.hep.tsinghua.edu.cn
```

从Windows登录:

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

# 上讲回顾(2)

---

## ■ Linux常用命令

帮助、查找相关：

man, find, grep, locate, ...

目录相关：

ls, cd, pwd, mkdir, rm, rmdir,...

复制、更名、压缩相关：

cp, mv, tar...

查看文件相关：

less, more, cat, head, tail,...

编辑文件相关：

vi, emacs, nano, pico,...

其它：

echo, sed, wc, history, du, chmod, chown, date, file,....

# 上讲回顾(3)

---

## ■ Shell、环境变量、脚本编程

脚本中定义变量、判断语句、循环语句...

PATH, HOME, PWD, USER, GROUP, ...

```
bash export MYDIR=/projects/yangzw/examples
```

```
tcsh setenv MYDIR /projects/yangzw/examples
```

## ■ Linux下的文本编辑器(emacs, vi, etc)

要学会熟练使用某种编辑器，不要把它们当成记事本使用，编程与一般的文档输入不同，强大的编辑器可以使编程变得高效。

# SHELL例子回顾

```
#!/bin/bash
```

```
# to run: ./ex1_45.sh 5
```

```
if [ ! $1 ]; then
```

```
    echo "para needed"; exit
```

```
fi
```

```
if [ ! -d /projects/$USER/try ]; then
```

```
    mkdir -p /projects/$USER/try
```

```
fi
```

```
export DR=/projects/$USER/try; cd $DR
```

```
#####for loop####
```

```
for i in `ls $DR`
```

```
do
```

```
    if [ -f $i ]; then chmod 744 $i; fi
```

```
done
```

```
#####while loop####
```

```
rm -f try*.html; n=1; N=$1
```

```
while (( $n < $N ))
```

```
do
```

```
    NAME=try$n.htm
```

```
    echo -e "$n:\nloop\nline3" > $NAME
```

```
    echo "line4..." >> $NAME
```

```
    sed -i "s/^/\n/g" $NAME
```

```
    let n+=1
```

```
done
```

```
rename .htm .html *.htm
```

```
echo "par0: $0 par1: $1"; date
```

指定**bash**，运行时需要加参数：**#!**开头

如果没有指定参数，则退出：**参数和分号的使用**

如果目录不存在，新建相应的目录。定义环境变量，进入该目录：**目录判断和mkdir -p的使用**

让变量*i*在**ls \$DR**的结果中循环，判断**\$i**是否为文件，若是则修改文件权限为用户可**rw**x，其他人只能**r**：**for**循环，文件判断，修改权限的使用

强制删除**try\*.html**文件。初始化变量**n**，**N**。让**n**从1循环到**N**，每次新建一个**.htm**文件，比如**try1.htm**。往文件中写入4行信息。(注：**\n**表示换行。)在文件所有行前面加上**"/"**(可用于**C/C++**程序的注释)。(注：**"/"**为特殊字符，在正规表达式中要用**"\"**进行转义，比如**"\"/\"/"**被解释为**"/"/**)：**while**循环，**\n**换行，**sed**用法，特殊字符和转义符的使用

将目录中所有**.htm**文件改名为**.html**文件，显示参数**\$0**，**\$1**和系统时间：**rename**，**date**的使用

注：脚本位于**/projects/yangzw/scripts/**

# 本讲摘要

## ■ 介绍C++与Linux下C++编译

g++, gmake

## ■ C++的基本概念

变量,类型与表达式

循环,类型设置,函数

文件与流程

数组,字符串,指针

类,面向对象设计介绍 (重点)

内存分配,算符使用与模版

继承关系,标准C++程序库,编译,查错

## ■ 如何进入并使用ROOT程序包

不系统介绍,  
在例子中学习使用  
C++关键概念  
练习后面附录了这些  
概念的简单介绍

# C++的历史简介

C++ 源自 C，最先由 Bjarne Stroustrup 于 80 年代早期完成 1998年正式标准化，成为今天的C++。

## ■ 全面兼容C

- 它保持了C的简洁、高效和接近汇编语言等特点
- 对C的类型系统进行了改革和扩充
- C++也支持面向过程的程序设计，不是一个纯正的面向对象的语言

## ■ 支持面向对象的方法

类(class)的概念：与C语言的最大区别！！

# 如何编译并执行一个C++程序

~yangzw/examples/Lec2/example21/HelloWorld.cc

首先用**emacs/vi**, 编写包含以下内容的文件 **HelloWorld.cc**

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

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

**提示符** **> ./HelloWorl**  
**Hello World!**

← 用户键入(注意: >为系统提示符)

← 计算机显示结果



# 在编译与链接中应注意的事项

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

是把编译与链接结合在一起的简化方式，是下列指令的组合

```
g++ -c HelloWorld.cc
```

编译器 (-c) 先产生 **HelloWorld.o** 目标文件，然后采用下面的指令链接目标文件

```
g++ -o HelloWorld HelloWorld.o
```

如果程序包含多个源文件，可将它们一一列出，并用空格区分；结尾用 \ 来起新的一行

```
g++ -o HelloWorld HelloWorld.cc Bonjour.cc \  
GruessGott.cc YoDude.cc
```

提示行中键入 `man g++` 可以查看 `g++` 命令的各个参数及用法

# 在编译与链接中还应注意的事项

1)通常给每一个程序产生一个新目录

**【所有的例子以及练习都保存好，以备将来查用，目录或者文件名字要尽量有意义，并做好记录，记录这些文件夹或文件的主要功能。该建议不限于C++程序，适用于所有例子和练习】学习的开始阶段做好笔记非常重要！**

2)对小的程序可以采用手工键入编译指令

3)对稍微大一些的程序项目，应采用写 shell 脚本的方法把编译程序所需的指令统统包含进去：

```
#!/bin/bash
# File build.sh to build HelloWorld
g++ -o HelloWorld HelloWorld.cc Bonjour.cc \
GruessGott.cc YoDude.cc
```

4) **更好的办法是使用GNU make 的 makefile!!!!!!**

# 如何编译并执行复杂的C++程序(1)

~yangzw/examples/Lec2/example22

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example22)，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)。

```
[training] /projects/yangzw/examples/Lec2/example22 > ls
bin      compile.sh  Makefile      Makefile.not.easy  src
build.sh include     Makefile.easy obj
```

# 如何编译并执行复杂的C++程序(2)

~yangzw/examples/Lec2/example22

请执行下面几个指令，复制文件、编译、运行C++的例子

**cd /projects/\$USER**

**cp -r ~yangzw/workdir/examples/Lec2/example22 .**

**./build.sh**      编译(确保有执行权限)

**./bin/try**      执行

不要忽略".", 当前目录

1)include目录存放头文件VolCuboid.h,

定义了名为VolCuboid的类

2)src目录存放源文件ex22.cc和VolCuboid.cc,

其中ex22.cc为主程序，它include了头文件VolCuboid.h

**#!/bin/sh**

build.sh的内容

**# File build.sh to build example22**

**g++ -o bin/ex22 -Iinclude/ src/\*.cc**

编译指令

输出可

执行文件

指定头文件目录

要编译的

源文件

# 如何编译并执行复杂的C++程序(3)

~yangzw/examples/Lec2/example22/Makefile

```
#Makefile: a simple makefile
default: hello

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

使用  
Makefile

1. makefile文件的名字必须为Makefile,makefile或GNUMakefile
2. 注释: 以#开头
3. 目标: hello, 该目标被指定为默认目标
4. 生成目标的命令: g++ -o bin/hello .....
5. 命令以Tab键开头!!!

```
>cd /projects/$USER/example22
```

```
>gmake clean    清除以前编译结果(不一定需要)
```

```
>gmake          编译
```

```
>bin/hello      运行
```

如果bin/hello存在, 且文档比源文件和头文件新, 再次提交

gmake命令系统不会重新编译, 节省时间

# 如何编译并执行复杂的C++程序(4)

~/yangzw/examples/Lec2/example22/Makefile.not.easy

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 +=
```

头文件或者库文件目录

g++ 命令的参数

```
# # set up compilers #
```

```
CPP = g++
```

```
CPPFLAGS = -O -Wall -fPIC -I$(INCLUDE)
```

可执行文件

```
##### Make Executables #####
```

```
all: ex22
```

```
ex22 : $(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)*
```

C++ 后缀,如所有.cc改为.cxx

备份原来的Makefile, 然后  
cp -a Makefile.not.easy Makefile  
gmake

# 如何编译并执行复杂的C++程序(5)

~yangzw/examples/Lec2/example22/Makefile.not.easy

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

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

```
# # set up compilers #
```

```
CPP = g++
```

```
CPPFLAGS = -O -Wall -fPIC -I$(INCLUDE)
```

```
##### Make Executables #####
```

```
all: ex22
```

```
ex22 : $(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)*
```

如果程序中调用ROOT程序包  
需要将ROOT的include和lib  
加入INCLUDE和CPPLIBS变量  
(红色和蓝色部分)

请在终端命令行执行下面命令，  
看看效果：

```
root-config --cflags
root-config --libs
root-config --glibs
```

# 语法解释(1)--基本概念

~yangzw/examples/Lec2/example23/CGrammar.cxx

```
//My first C++ program
//注释: 好的注释的程序的重要部分
//头文件, iostream负责输入输出
//提供cout, cin等基本函数
#include <iostream>
//使用std名字空间
using namespace std;

//主函数, 返回整型值。必须
int main(){
    //屏幕输出Hello字符串和换行符
    std::cout << "Hello"
               << endl;
    //整型、浮点型、双精度变量
    int Nevt=20;
    float px=10.0;
    double mass=3.1;
```

```
//for循环
for (int i=0;i<10;i++){
    Nevt++;
    //if判断语句
    if(Nevt>15){
        cout << "Nevt: "
              << Nevt
              << std::endl;
    }//end of if
}//end of for i<10

//为主函数返回整型值
return 0;
}
注:
函数或者循环语句要放在大括号{}
里。
```



# 语法解释(2)--类的定义

~yangzw/examples/Lec2/example22/include/VolCuboid.h

```
//#####类VolCuboid#####  
//#####计算体积#####  
#ifndef VOLCUBOID_H //防止重复引用  
#define VOLCUBOID_H  
  
#include <iostream>  
  
class VolCuboid { //类的名称  
    //公共的变量或函数，可以外部调用  
public:  
    VolCuboid(float x, float y, float z); //构造函数  
    ~VolCuboid(); //析构函数，删除动态指针、成员函数等  
    float Vol(); //成员函数，实现某种功能  
    //私有的变量或函数，供类内部使用  
private:  
    float length, width, height;  
}; //注意这里有个分号  
  
#endif
```

2011-3-10

# 语法解释(3)--类的函数

~yangzw/examples/Lec2/example22/src/VolCuboid.cc

//引用类的定义

```
#include "VolCuboid.h"
```

//构造函数**VolCuboid**，必须与类的名字一样。

//需要立方体长宽高三个参数

```
VolCuboid::VolCuboid(float x, float y, float z) {
```

```
    length = x ;
```

```
    width  = y ;
```

```
    height = z ;
```

```
}
```

//析构函数**~VolCuboid()**，啥也没做。

```
VolCuboid::~~VolCuboid() {
```

```
    //new pointers should be deleted here.
```

```
    //if not, do nothing.
```

```
}
```

//成员函数**vol()**，计算并返回立方体体积。

```
float VolCuboid::Vol() {
```

```
    return length*width*height;
```

```
}
```

用类定义对象：

**VolCuboid myCuboid(3,4,5);**

或者 **VolCuboid \*myCuboid = new VolCuboid(3,4,5);**

# 语法解释(4)--主程序

~yangzw/examples/Lec2/example22/src/ex22.cc

主程序：用类VolCuboid计算体积

```
///#####ex22.cc#####
```

```
#include <iostream>
```

```
#include "VolCuboid.h"
```

include定义类VolCuboid的头文件

```
using namespace std;
```

定义立方体长宽高

```
int main () {
```

```
    std::cout << "exercise 22..." << endl;
```

```
    float length, width, height;
```

```
    length = 2.0 ; //cm
```

```
    width = 3.0 ; //cm
```

```
    height = 4.0 ; //cm
```

用类VolCuboid构造其对象

```
    VolCuboid myVolCuboid( length, width, height );
```

调用成员函数Vol()

```
    float volume = myVolCuboid.Vol() ;
```

计算体积。

```
    cout << "Volume:" << volume << endl;
```

```
}
```

注：若用new指针方式构造对象：VolCuboid \*myCuboid = new VolCuboid(3,4,5);  
则函数调用需要用"->"而不是"."：float volume = myVolCuboid->Vol();

# 运行ROOT程序包

ROOT是有CERN开发的数据分析处理软件包，兼容C++语法，本身带cint编译器，可以自动编译执行C++代码。多数情况下，将C++的命令行直接放到大括号{ }中就可以在ROOT环境下运行

ROOT安装在\$ROOTSYS目录。

进入ROOT环境：root 或者root -l

退出ROOT环境：.q

运行程序段如hello.C: root -l hello.C

官网： <http://root.cern.ch>

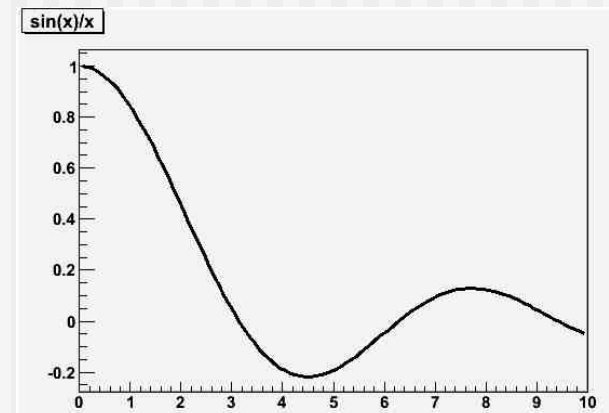
Windows版本下载，双击安装即可

[ftp://root.cern.ch/root/root\\_v5.28.00a.win32.msi](ftp://root.cern.ch/root/root_v5.28.00a.win32.msi)

# ROOT包Macro程序段解释(1)

~yangzw/examples/Lec2/example24/hello.C

```
{ //脚本放到大括号里面(只是方式之一)
  cout << "Welcome to ROOT." << endl;
  int Num = 10;
  float x = 0.2;
  cout << "Num = " << Num << endl;
  cout << "x  = " <<  x << endl;
  //定义函数sin(x)/x, 区间0-10
  TF1 f1("func1","sin(x)/x",0,10);
  f1.Draw(); //画出函数f1
}
```

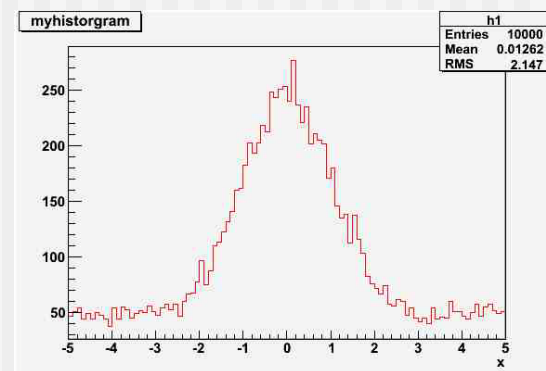


1. 后缀名一般为.C
2. 运行: root -l hello.C 或者进入root环境后执行  
.x hello.C (注意, 要先进入root环境)

# ROOT包Macro程序段解释(2)

~yangzw/examples/Lec2/example24/hello1.C

```
{  
    //定义一维直方图(浮点型)  
    // 定义直方图需要的参数: 名字      描述      区间数  上下限  
    TH1F *h1 = new TH1F("h1","myhistogram", 100, -5., 5.);  
    //用标准高斯分布填充直方图5000次  
    h1->FillRandom("gaus",5000);  
    for (int i=0;i<1000;i++) {  
        //产生[-5,5]区间均匀分布的随机数,并填充到直方图中  
        float x=gRandom->Uniform(-5.,5.);  
        h1->Fill(x);  
    }  
    h1->SetLineColor(2);  
    h1->GetXaxis()->SetTitle("x");  
    h1->Draw(); //画出直方图  
}
```



运行root -l hello1.C可以得到结果

# 小结

---

- Shell脚本编程回顾
- g++编译C++程序
- 用Makefile编译C++程序
- C++基本概念：变量，循环和类
- ROOT脚本(macro)的运行

# 练习

1. 定义类Cuboid(float x, float y, float z), 类中定义3个成员函数, 分别用于计算长方体体积、表面积和棱长。类定义完之后, 写个主程序ex2\_1.cxx, 使用刚定义的类计算给定长宽高(3,4,5)的长方体的体积、表面积和棱长, 显示结果于屏幕上。  
(提示: 在讲义例题的基础上修改)
2. 练习用gmake编译该程序以及g++编译, 分别写个脚本和Makefile编译例题中的CGrammar.cxx
3. 将练习1备份好之后, 在主程序中引用头文件TMath.h, 并在屏幕打印圆周率Pi以及自然常数的值。  
(提示: TMath.h为ROOT程序包的头文件之一, 其中定义了各种数学和物理常数的函数, 这些函数都放在名字空间TMath下。该文件的目录为\$ROOTSYS/include/TMath.h)
4. 将讲义例题CGrammar.cxx中#if 0一行中的0换为1, 查看并修改数组以及const常量的用法。
5. 熟悉C++的其它关键概念, 比如数组、指针等。
6. 阅读ROOT使用手册的第二章, 熟悉基本命令和规则



# 参考资料

---

1. C++ Primer中文版, S. Lippman著, 潘爱民译, 人民邮电出版社
2. C++大学自学教程, A. Stevens著, 林瑶等译, 电子工业出版社
3. Linux环境下C编程指南, 杨树青、王欢编著, 清华大学出版社
4. [http://www.linuxsir.org/main/doc/gnumake/GNUMake\\_v3.80-zh\\_CN\\_html/index.html](http://www.linuxsir.org/main/doc/gnumake/GNUMake_v3.80-zh_CN_html/index.html)

这是GNUMake中文手册

5. <http://root.cern.ch>

其中的UsersGuide, 以及tutorials, howtos

UsersGuide 5.16版本可在此处下载

<http://hep.tsinghua.edu.cn/~yangzw/CourseDataAna/refs/ROOT/>

# HelloWorld.cc 程序详解

// My first C++ program      注释行

过去沿用的“C 标准”仍然可以使用：

```
/*
```

    注释行

    放在它们之间

```
*/
```

```
/* 注释行还可以放在它们之间 */
```

为了您日后或他人理解您的程序与思路，请在程序中加上足够的注释行。

每个文件应在文件开始的注释行中注明作者姓名，该程序的目的，所需要的外部输入信息，等等。

# HelloWorld.cc 程序中的包含语句

`#include <iostream>` 是一个给编译器的指令

给编译器的指令以“#”开头。这种语句在程序运行当中不会被执行，而是给编译器提供信息。

`#include <iostream>` 告诉编译器程序将使用库文件，文件的定义可以在一个叫 `iostream` 的文件中找到，通常情况下该文件放在 `/usr/include`。注意在以往编程中，通常将其写为 `#include <iostream.h>`。

`iostream` 包括了执行与键盘和显示器通讯的各种 `i/o` 操作的函数。

在本程序中，我们采用 `iostream` 的目标模块 `cout` 来传送文字到显示器。该指令在大多数程序中使用。

# HelloWorld.cc 程序中的其它语句

`using namespace std;` 以后再详述。

一个 C++ 程序由一系列函数组成。每个程序只能包含一个叫 `main` 的主函数：

```
int main(){  
    // 将程序的主体放在此  
  
    return 0;  
}
```

函数中返回 “`return`” 一个给定类型的值；主函数 `main` 返回 `int` (整数)。

`()` 用于可能的变量输入，这里的主程序 `main` 无输入量。

函数的主体以大括号表示：{ }

`return 0;` 表示主函数 `main` 返回 0 值。

# HelloWorld.cc 中的输出语句

HelloWorld.cc 要实现的功能包含在下面一行

```
cout << "Hello World!" << endl;
```

与其它语句一样，结束时需采用分号结尾。

**cout** 是输出流目标模块(output stream object)。

用 **<<** 把双引号中的字符串送给 **cout**

它还可以将数值自动转换为字符串输出，例如

```
cout << "x = " << x << endl;
```

把 **endl** 送给 **cout** 表明应新起一行。

过去也采用 “Hello World!\n” 来表示换行。

在调试大性程序时，经常使用 **cout** 来检查错误。

# C++ 的组成要素

在一个 C++ 程序中：

**保留词：**是不可改变的，例如 `if, else, int, double, for, while, class, ...`

**库函数标识：**缺省含义通常不可改变， 例如

`cout, sqrt (开根号), ...`

**编程者提供的标识，**例如变量名

`x, y, probeTemperature, photonEnergy, ...`

有效的标识必须以字母或下划线( “  ” )开头，可以包含字母

、

数字以及下画线。最好使用可以辨认的英文单词来命名标识。注意名字的大小字母是有区别的。

# 数据类型

---

数据的值可以储存在几种类型的变量中：

**基本整型类型**：`int` (以及 `short`, `unsigned`, `long int`, ...)

字节长度取决于所采用的编译器；通常是 32 位。

**基本浮点类型** (例如对实数而言)：

`float`      通常是 32 位

`double`    通常是 64 位

**逻辑类型**(boolean): `bool` (等于 `true` 或 `false`)

**字符类型**: `char` (单个 ASCII 字符而已, 也可以为空格符),  
不是真正意义上的字符类。

**枚举类型**: `enum`(用标志符来表示整数)。

# 定义变量名

所有变量在使用前必须事先声明。 声明的两种方式：

在程序开头定义 (象 FORTRAN);

仅在第一次使用前定义 (与 java 一样)。 ← 建议采用

例子

```
int main(){
    int numPhotons;           // Use int to count things
    double photonEnergy;      // Use double for reals
    bool goodEvent;           // Use bool for true or false
    int minNum, maxNum;        // More than one on line
    int n = 17;                // Can initialize value
    double x = 37.2;           // when variable declared.
    char yesOrNo = 'y';        // Value of char in ' '
    ...
}
```



# 给变量赋值

定义变量只是给其建了一个名字，并没有赋值(除非在定义变量时还一起赋值)。赋值采用“=”号来完成。

例子

```
int main(){
    bool aOK = true;    // true, predefined constants
    double x, y, z;
    x = 3.7;
    y = 5.2;
    z = x + y;
    cout << "z = " << z << endl;
    z = z + 2.8;         // N.B. not like usual equation
    cout << "now z = " << z << endl;
    ...
}
```

# 常数

---

我们有时要保证一个变量的值不变

最好把参数放在易于发现之处，以便于日后做可能的修改  
用关键词 **const** 来定义：

例子

```
const int numChannels = 12;
const double PI = 3.14159265;

// Attempted redefinition by
// Indiana State Legislature
PI = 3.2;           // ERROR will not compile
```

还保留以往的 C 格式中如下的写法以便具有兼容性：

```
#define PI 3.14159265
```

# 枚举

---

有时我们想把数值赋给单词，例如

January = 1, February = 2, 等等。

可以采用关键词 `enum` 进行枚举处理

```
enum { RED, GREEN, BLUE };
```

是下面的简写方式

```
const int RED = 0;  
const int GREEN = 1;  
const int BLUE = 2;
```

枚举以缺省零值开始，可以改变

```
enum { RED = 1, GREEN = 3, BLUE = 7 }
```

(若不写明数值，则它们的值将在原来的基础上加一)

# 表达式

C++ 由其特征的数学表达式符号：

功能	符号
加	+
减	-
乘	*
除	/
模数	%

注意：整型量 `int` 相除的结果是截断值

```
int n, m;  n = 5;  m = 3;  
int ratio = n/m;           // ratio has value of 1
```

模数给出整型量相除后的余值：

```
int nModM = n%m;           // nModM has value 2
```

# 算符执行的优先级

\* 与 / 优先级比 + 与 - 高, 例如

$x*y + u/v$  意味着  $(x*y) + (u/v)$

\* 与 / 有相同的优先级, 执行起来是从左到右:

$x/y/u*v$  意味着  $((x/y) / u) * v$

类似的法规适用于 + 和 -

$x - y + z$  意味着  $(x - y) + z$


由于优先级不是非常好记, 写程序时, 建议用括号来告诉计算机如何处理算符执行的优先级。

# 逻辑运算与表达式

逻辑表达式是 true 或 false, 例如

```
int n, m; n = 5; m = 3;  
bool b = n < m;           // value of b is false
```

C++ 中的逻辑表达式符号:

greater than	>	
greater than or equals	>=	
less than	<	
less than or equals	<=	
equals	==	 不是 =
not equals	!=	

可以用 **&&** ( “与” ), **||** ( “或” ) 和 **!** ( “非” ), 例如

```
(n < m) && (n != 0)           (false)  
(n%m >= 5) || !(n == m)      (true)
```

逻辑算符的优先级不是很明显, 如有不确定性应采用括号。

# 简化的语句

## 完整的语句

## 简化的语句

$n = n - m$

$n -= m$

$n = n * m$

$n *= m$

$n = n / m$

$n /= m$

$n = n \% m$

$n \% = m$

在只递增或递减一的特殊情况下：

## 完整的语句

## 简化的语句

$n = n + 1$

$n++$  (或  $++n$ )

$n = n - 1$

$n--$  (或  $--n$ )

变量前的 $++$  或  $--$  意味着先递增 (或递减) 然后再执行语句中其它操作。

# 从键盘输入取值

有时我们要从键盘中键入一个值并赋给一个变量，这种情况可使用 `iostream` 目标模块 `cin`:

```
int age;  
cout << "Enter your age" << endl;  
cin >> age;  
cout << "Your age is " << age << endl;
```

当运行程序是，会有下列反应

Enter your age

23

← 在此键入数值，然后回车

Your age is 23



# if 与 else

程序流程控制可通过 if 和 else 来实现:

```
if ( 逻辑表达式 ){  
    如果通过检验则执行的语句  
}  
  
或  
  
if ( 表达式 1 ){  
    如果表达式 1 为真则执行的语句  
}  
else if ( 表达式 2 ) {  
    如果表达式 1 为假但表达式 2 为真则执行的语句  
}  
else {  
    如果表达式 1 与表达式 2 皆为假则执行的语句  
}
```

# 写成完整的程序

---

```
#include <iostream>
using namespace std;
int main() {
    const double maxArea = 20.0;
    double width, height;
    cout << "Enter width" << endl;
    cin >> width;
    cout << "Enter height" << endl;
    cin >> height;
    double area = width*height;
    if ( area > maxArea ){
        cout << "Area too large" << endl;
    else {
        cout << "Dimensions are OK" << endl;
    }}
    return 0;}
```

# 循环语句 while

一个 **while** 循环允许重复一系列语句，只要特定的条件仍然为真：

```
while( 逻辑表达式 ){  
    ...  
}
```

下面的情况也常用到，即每次循环通过一次后，循环中的逻辑表达式必须被重新检查一次：

```
while (x < xMax){  
    x += y;  
    ...  
}
```

这种处理可以避免**死循环**。

# 循环语句 do-while

---

一个 **do-while** 循环类似于一个 **while** 循环，但是它总是执行至少一次，然后继续直到特定的逻辑条件不再满足为止。

```
do {  
    ...  
} while ( 逻辑表达式 )
```

需要留意在第一次循环时，逻辑表达可能发生改变，需要进行必要初始化。

# 循环语句 for

一个 **for** 循环允许有限次重复一系列语句，具有下列形式

```
for ( 初始化操作 ;  
      逻辑表达式 ; 更新操作 ){  
    ...  
}
```

下列方式最普遍：

```
for (int i=0; i<n; i++){  
    ...  
}
```

注意这里 **i** 仅定义在括号 **()** 里。

# 循环语句举例

---

一个 **for** 循环:

```
int sum = 0;
for (int i = 1; i<=n; i++){
    sum += i;
}
cout << "sum of integers from 1 to " << n <<
      " is " << sum << endl;
```

一个 **do-while** 循环:

```
int n;
bool gotValidInput = false;
do {
    cout << "Enter a positive integer" << endl;
    cin >> n;
    gotValidInput = n > 0;
} while ( !gotValidInput );
```

# 多重循环

---

可以利用循环语句进行有目的的多次循环，例如

```
// loop over pixels in an image

for (int row=1; row<=nRows; row++){
    for (int column=1; column<=nColumns; column++){
        int b = imageBrightness(row, column);
        ...
    }        // loop over columns ends here
}           // loop over rows ends here
```

可以把任何类型的循环语句嵌入到另一类型的循环语句中，例如 **while** 循环可嵌入**for** 循环中，反之亦然。

# 可控制的循环

**continue** 跳过一次循环回到循环起点

**break** 退出整个循环(注意对于多重循环只退出最里层)

```
while ( processEvent ) {  
  
    if ( eventSize > maxSize ) { continue; }  
  
    if ( numEventsDone > maxEventsDone ) {  
        break;  
    }  
  
    // rest of statements in loop ...  
  
}
```

实际应用中最好避免采用 **continue** 或 **break**, 可以使用 **if** 语句来代替。



# 类型转换

假设有: `int n, m; n = 5; m = 3;` 我们想知道真实没有截断的 `n/m` 值。这个时候, 我们需要进行类型转换把 `n` 与 `m` 从整型 `int` 转换到双精度 `double` (或浮点 `float`):

```
double x = static_cast<double>(n) /  
           static_cast<double>(m);
```

将得到 `x = 1.666666...`

也可采用 `static_cast<double>(n)/m;` 得到同样的结果。但是 `static_cast<double>(n/m);` 会给出 `1.0` 的结果。

类似地, 采用 `static_cast<int>(x)` 把一个 `float` 或 `double` 转换为 `int`, 等等。

# 标准数学函数

简单的数学函数可直接调用标准的 C 语言库 `cmath` (过去为 `math.h`)。它包括：

```
abs    acos    asin    atan    atan2    cos    cosh    exp  
fabs    fmod    log     log10    pow     sin     sinh    sqrt  
tan     tanh
```

它们中许多是可以用 `float` 或 `double` 输入量的，而返回值为同一数据类型。

# 一个简单的例子

例如产生下列文件 `testMath.cc` 包含:

```
// Simple program to illustrate cmath library
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    for (int i=1; i<=10; i++){
        double x = static_cast<double>(i);
        double y = sqrt(x);
        double z = pow(x, 1./3.);    // note decimal pts
        cout << x << " " << y << " " << z << endl;
    }
}
```

编译链接: `g++ -o testMath testMath.cc`

运行并输出到文件result: `./testMath > result`

# 格式输出

通常情况下，一般要控制数值输出的精度，例如

```
cout.setf(ios::fixed);  
cout.precision(4);
```

将给出小数点后四位有效数字。

```
cout.setf(ios::scientific);
```

将给出以科学计数法表示的结果，例如，**3.4516e+05**。如果需要取消该设置，可以通过下列方式来实现

```
use cout.unsetf(ios::scientific);
```

如果要输出空格符，可按下列方式进行

```
cout.width(15);
```

将给出15个空格符，例如

```
cout.width(5); cout << x;  
cout.width(10); cout << y;<< endl;
```

注意使用 **cout.width** 需要加入 **#include <iomanip>** 。

# 指令 printf 与 scanf 的使用

在格式输出中还可以采用 C 语言函数 `printf`:

```
printf ("formatting info" [, arguments]);
```

例如, 对于 `float` 或 `double x` 和 `int i`:

```
printf("%f %d \n", x, i);
```

将给出 `x` 的值含小数点后有效数字, 以及 `i` 的整数值。这里换行指令 `\n` 与 `endl` 一样;

如果我们要给 `x` 8 位的空间, 其中 3 位为小数点后, 以及 10 位空间给 `i`:

```
printf("%8.3f %10d \n", x, i);
```

指令 `scanf` 可参照 `cin` 的使用方法。

使用 `printf` 与 `scanf` 需要加入 `#include <cstdio>` 。

# 类型定义的使用范围

特别注意在变量类型定义中，如果定义在`{}`之内，该定义将无法在其作用域之外使用，例如

```
int x = 5;
for (int i=0; i<n; i++){
    int y = i + 3;
    x = x + y;
}
cout << "x = " << x << endl; // OK
cout << "y = " << y << endl; // BUG: y 出了作用域
cout << "i = " << i << endl; // BUG: i 出了作用域
```

如果想让变量类型适用于全局，应在所用函数包括 `main` 外进行所谓的全局变量定义。

# 函数

用户可以根据需要定义自己的函数

```
const double PI = 3.14159265;    // 全局常数
double ellipseArea(double, double); // 原型
int main() {
    double a = 5;  double b = 7; double height = 10;
    double area = ellipseArea(a, b);
    cout << "area = " << area << endl;
    double volume = ellipseArea(a, b)*height;
    cout << "volume = " << volume << endl;
    return 0;
}
double ellipseArea(double a, double b){
    return PI*a*b;
}
```

在 **main** 之前，应定义函数输入变量和函数返还值的类型。

# 函数返回类型 void

---

函数返还类型可以为无 ‘**void**’, 这种情况表明无返还声明, 类似于 FORTRAN 语言的子程序:

```
void showProduct(double a, double b){  
    cout << "a*b = " << a*b << endl;  
}
```

为了调用具有 void 空的函数, 可以简单采用下面的方式:

```
showProduct(3, 7);
```



# 把函数放在分开的文件中

通常我们把函数的代码放在分开的文件里是为了便于管理与使用。这时函数的声明要放在头文件里 ‘header file’。例如下例中的头文件 `ellipseArea.h`，注意给每个函数加上注释行说明用途

```
#ifndef ELLIPSE_AREA_H
#define ELLIPSE_AREA_H

// function to compute area of an ellipse

double ellipseArea(double, double);

#endif
```

指向 `#ifndef` (如果没有定义) 用于保证函数的原型不被多次重复定义。如果 `ELLIPSE_AREA_H` 已经被声明过，则跳过声明区。

# 在main与函数程序中的头文件

---

在函数程序中，应包括头文件名

```
#include "ellipseArea.h"
double ellipseArea(double a, double b){
    return PI*a*b;
}
```

在主程序main中，也同样应包括头文件名

```
#include "ellipseArea.h"
int main() {
    double a = 5; double b = 7;
    double area = ellipseArea(a, b);
    ...
}
```

# 函数输入量的变化与引用

如果希望函数可以**改变**输入变量的大小，例如一函数

```
void tryToChangeArg(int x){x = 2*x;}
```

并不能通过下列方式来改变输入变量的值：

```
int x = 1;
```

```
tryToChangeArg(x);
```

```
cout << "now x = " << x << endl; // x still = 1
```

输入变量值与函数内计算无关，在外部看来始终保持不变。

如果希望函数**可以改变**输入变量的大小，则采用 **&** 标记

```
void tryToChangeArg(int&); // 原型
```

```
void tryToChangeArg(int& x){x = 2*x;}
```

```
int main(){
```

```
    int x = 1;
```

↑  
引用变量

```
    tryToChangeArg(x);
```

```
    cout << "now x = " << x << endl; // 现在 x = 2
```

```
}
```

但是此时输入量必须是变量。 **tryToChangeArg(7);** 有误。

# 函数的重载

有时我们希望同一函数既可以处理 `float` 又可以处理 `double`  
称为**函数的重载**

```
double cube(double);  
double cube (double x){  
    return x*x*x;  
}  
double cube(float);  
double cube (float x){  
    double xd = static_cast<double>(x);  
    return xd*xd*xd;  
}
```

**C 标准数学库中的  
函数都进行了重载。**

```
float x;  
double y;  
double z = cube(x); // calls cube(float) version  
double z = cube(y); // calls cube(double) version
```

# 打开要读取的文件

---

我们经常要读入诸如数据等文件，并把数据处理结果写入文件中，例如

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(){
    // 生成一个 ifstream 目标模块 (名字任意)...
    ifstream myInput;
    // 打开一个已经存在的文件
    myInput.open("myDataFile.dat");
    // 检查上述操作是否正常
    if ( myInput.fail() ) {
        cout << "Sorry, couldn't open file" << endl;
        exit(1);          // from cstdlib
    }
}
```

# 读入数据流

输入文件流目标模块与 `cin` 类似,但是它不是从键盘中得到输入的数据,而是从文件中读入数据。注意使用点“.”来调用 `ifstream` 的成员函数,如 `open`, `fail`, 等等。假设文件为

```
1.0    7.38  0.43
2.0    8.59  0.52
3.0    9.01  0.55
...
```

通过下例方式把数据从文件中读出:

```
float x, y, z;
for(int i=1; i<=numLines; i++){
    myInput >> x >> y >> z;
    cout << "Read " << x << " " << y << " " << z << endl;
}
```

注意该循环需要知道文件中有多少行 `numLines`。

## 另一种读入数据流的方式

通常情况下，我们不知道文件有多少行，此时，可以使用所谓的文件结尾 (**eof**) 函数：

```
float x, y, z;
int line = 0;
while ( !myInput.eof() ){
    myInput >> x >> y >> z;
    if ( !myInput.eof() ) {
        line++;
        cout << x << " " << y << " " << z << endl;
    }
    cout << line << " lines read from file" << endl;
}
...
myInput.close();          // close when finished
```

# 打开要写入的文件

我们可以利用 `ofstream` 目标模块把数据写入文件中：

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(){
    // 生成一个 ofstream 目标模块 (名字任意)...
    ofstream myOutput;
    // 打开一个新文件
    myOutput.open("myDataFile.dat");
    // 检查上述操作是否正常
    if ( myOutput.fail() ) {
        cout << "Sorry, couldn't open file" << endl;
        exit(1);          // from cstdlib
    }
    ...
}
```

注意： `ifstream` 与 `ofstream`



# 把数据写入文件中

此时的 `ofstream` 目标模块运作起来像 `cout` 一样：

```
for (int i=1; i<=n; i++){  
    myOutput << i << "\t" << i*i << endl;  
}
```



表示跳格键

还可以采用函数 `setf`, `precision`, `width`, 等等, 控制输出格式。方法与 `cout` 使用一样

```
myOutput.setf(ios::fixed);  
myOutput.precision(4);  
...
```

# 文件的输出模式

在前面的例子中，程序将更新已存在文件中的内容。如果要把数据**接到文件的尾部**，可以通过下列方式指定：

```
myOutput.open("myDataFile.dat", ios::app);
```

如果要把数据以**二进制格式**写到文件中，以达到减小文件的大小的目的，可以采用：

```
myOutput.open("myDataFile.dat", ios::bin);
```

可以采用 “|” 来实现多个选项，例如

```
myOutput.open("myDataFile.dat", ios::bin|ios::app);
```

表明将数据以二进制格式写到文件尾部。

# 一个完整的读写例子

---

文件名 `examAve.cc`

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "aveScore.h"
using namespace std;
int main(){
    // open input file
    ifstream inFile;
    inFile.open("studentScores.dat");
    if ( inFile.fail() ) {
        cerr << "Couldn't open input file" << endl;
        exit(1);
    }
    ...
}
```

# 一个完整的读写例子（续一）

---

```
// open the output file
ofstream outFile;
outFile.open("averageScores.dat");
if ( outFile.fail() ) {
    cerr << "Couldn't open output file" << endl;
    exit(1);
}

while ( !inFile.eof() ){
    int studentNum;
    double test1, test2, test3;
    inFile >> studentNum >> test1 >> test2 >> test3;
    if( !inFile.eof() ){
        double ave = aveScore (test1, test2, test3);
        outFile << studentNum << "\t" << ave << endl;
    }
}
```

## 一个完整的读写例子（续二）

---

```
// close up
inFile.close();
outFile.close();
return 0;
}
```

而程序 `aveScore.cc` 包含

```
include "aveScore.h"
double aveScore(double a, double b, double c){
    double ave = (a + b + c)/3.0;
    return ave;
}
```

# 一个完整的读写例子（续三）

头文件 `aveScore.h` 包含

```
#ifndef AVE_SCORE_H
#define AVE_SCORE_H
double aveScore(double, double, double);
#endif AVE_SCORE_H
```

采用下列方式对程序进行编译与链接

```
g++ -o examAve examAve.cc aveScore.cc
```

输入的数据文件 `studentScores.dat` 可以为

1	73	65	68
2	52	45	44
3	83	85	91

# 数组

数组是固定长度并包含同类型变量的列表。应采用下列方式来定义 **(一维) 数组**：

*data-type variableName[numElements];*

例如

↑  
也称作数组的长度

```
int score[10];  
double energy[50], momentum[50];  
const int MaxParticles = 100;  
double ionizationRate[MaxParticles];
```

注意，数组的编号从 0 开始，例如 score[10]  
*score[0], score[1], score[2], ..., score[9]*

调用时不能出现 score[10]，否则会出现**数组超界错误**。

如果是**多维数组**，则是

*data-type variableName[num1][num2];*

# 数组的初始化

---

例如对于一维数组

```
int myArray[5] = {2, 4, 6, 8, 10};
```

二维数组

```
double matrix[numRows][numColumns] =  
    { {3, 7, 2}, {2, 5, 4} };
```



# 一个矩阵与矢量相乘的例子

---

```
// Initialize vector x and matrix A
const int nDim = 5;
double x[nDim];
double A[nDim][nDim];
for (int i=0; i<nDim; i++){
    x[i] = someFunction(i);
    for (int j=0; j<nDim; j++){
        A[i][j] = anotherFunction(i, j);
    }}

// Now find y = Ax
double y[nDim];
for (int i=0; i<nDim; i++){
    y[i] = 0.0;
    for (int j=0; j<nDim; j++){
        y[i] += A[i][j] * x[j];
    }}
}}
```

# 把一维数组传递给函数

```
double sumElements(double a[], int len);
```

↑  
数组的长度

在函数中不必定义具体的数组长度，通常在外部调用时定义，例如

```
double sumElements(double a[], int len){  
    double sum = 0.0;  
    for (int i=0; i<len; i++){  
        sum += a[i];  
    }  
    return sum;}  
注意调用时不用写[]
```

调用时

↓  

```
double s = sumElements(myMatrix, itsLength);
```

如果只想传递数组的一个元素到函数，可以采用下例方式

```
double s = sqrt(myMatrix[1]);
```

# 把多维数组传递给函数

---

多维数组的写法要求指明最左边的标识

```
void processImage(int image[][numColumns],  
                  int numRows, int numColumns){  
    ...
```

注意数组在函数内部数值的改变可以带到函数外部，与 & 标识作用一样。

# 指针、& 操作符与变量

一个指针变量包含了内存地址，它指向了内存的具体位置，其定义采用 “\*” 来完成，例如

```
int* iPtr;  
double * xPtr;  
char *c;  
float *x, *y;
```

注意：这里的指针变量均没有初始化，它们指向内存中的随机位置

注意，在

```
int* iPtr, jptr;
```

↑  
指针变量

↑  
非指针变量



```
int** iPtr, jptr;
```

如果要定义指针变量到包含变量 **i** 的内存具体位置，可以

```
int i = 3;  
int* iPtr = &i;
```

注意：这里的 & 指地址与前面把可变数值的变量传递到函数有本质的不同

也可把在内存某一位置的变量通过指针拷贝到变量 **i**，即

```
int iCopy = *iPtr;    // now iCopy equals i
```

# 把指针当成输入量传递

当指针被当作输入量进行传递时，它把相应的地址告诉了所调用的函数。因此，函数可以改变存在对应内存地址的变量值，例如：

```
void passPointer(int* iPtr){
    *iPtr += 2;           // note *iPtr on left!
}

...
int i = 3;
int* iPtr = &i;
passPointer(iPtr);
cout << "i = " << i << endl;    // prints i = 5
passPointer(&i);                // equivalent to above
cout << "i = " << i << endl;    // prints i = 7
```

# 指针与引用&

一个引用变量如同一个普通变量的别名，可通过在类型后加上 **&** 来表示：

```
void passReference(int& i){  
    ...  
}
```

而指针除了具有上述功能以外，还具有**动态地分配地址**：

```
double* array;
```

```
int len;
```

```
cout << "Enter array length" << endl;
```

```
cin >> len;           // 数组长度可以动态改变
```

```
array = new double[len];
```

```
...
```

比较



```
int len=10; //固定  
double array[len];
```

但是在完成使用以后，需要把该动态数组删除：

```
delete [] array;
```

在以后使用 ROOT 数据分析  
软件包时，常用到该操作。

# 字符串

字符串可以由类型为 **char** 的数组来表示，例如

```
char aString[] = "hello";
```

程序库 **cstring** ( **#include <cstring>** ) 提供了拷贝、并合与寻找子字符串等函数，例如

```
char* strcpy(char* target, const char* source);
```

它完成输入字符串 **source** 并令字符串 **target** 等于它。注意 **source** 被当作 **const** 来传递，不能改变。

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
```

```
    char string1[] = "hello"; char string2[50];
    strcpy(string2, string1);
    cout << "string2:  " << string2 << endl;
    return 0;}
```

2011-3-10