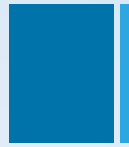




C++

第二讲 C++对C的改进（一）

C++备课组 丁盟



自我介绍

丁盟

qq : 2622885094





上一讲教学目标

- 了解C++的起源
- 了解C++的应用领域
- 理解对象和类的基本概念
- 理解面向对象的三大特征
- 安装VS集成开发环境

本讲教学目标

- 理解赋值和初始化的区别
- 了解C++中的注释的写法
- 了解C++中的输入输出的写法
- 理解命名空间的概念
- 了解C++中的数据类型
- 掌握C++中const的使用

1

C与C++的基本区别

新的初始化方法

新的I/O

新的注释

名字空间

有关类型的区别

2

独具魅力的const修饰符

新的初始化方法



赋值：更改变量的值



**初始化：定义变量的
同时进行付初始值**

新的初始化方法

❖ C提供的初始化方法

```
int x = 1024;
```

创建一个名为x
的数据对象，该
对象的初值为
1024

❖ C++提供两种初始化方法

复制初始化(copy-initialization)

如：int x = 1024;

直接初始化(direct-initialization)：

如：int x(1024);

新的初始化方法

❖[注意]

- ① 初始化不是简单地赋值，初始化指声明变量或对象并且赋初值；赋值指用新值覆盖变量或对象当前值。
- ② 直接初始化语法更灵活且效率更高
- ③ 初始化内置类型变量两种初始化几乎没有差别对于类类型的初始化，有时只能采用直接初始化（以后讨论）
- ④ 两种初始化的方法可以混用(见下页例)

新的初始化方法

❖ 混合使用初始化的例子:

```
#include <iostream>
using namespace std;
int main(void)
{
    double salary = 9999.99, wage(salary + 0.01);
    cout<<salary<<" "<<wage<<endl;

    int interval, month = 8, day = 7, year(2008);
    cout<<year<<":"<<month<<":"<<day<<":"<<endl;

    return 0;
}
```

变量的定义随时
定义随时使用

1

C与C++的基本区别

新的初始化方法

新的I/O

新的注释

名字空间

有关类型的区别

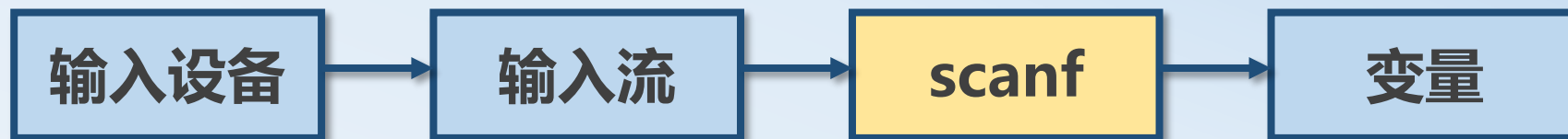
2

独具魅力的const修饰符

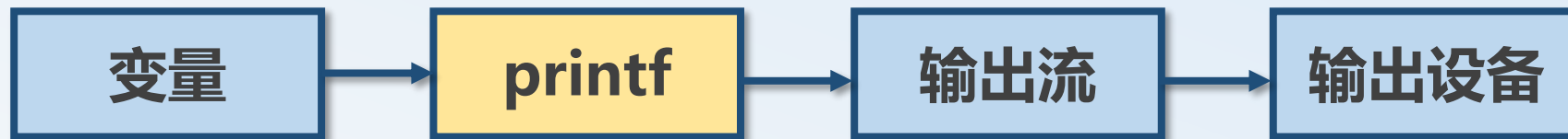
新的I/O

❖ C语言的输入输出

➤ 输入过程



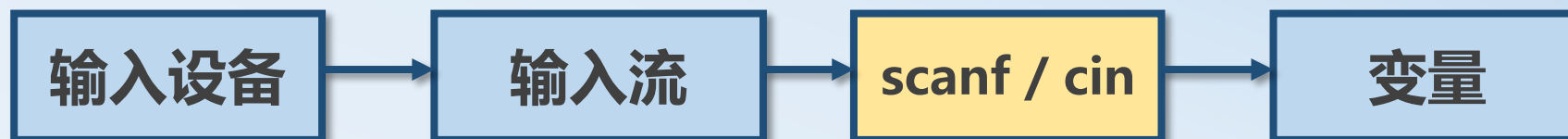
➤ 输出过程



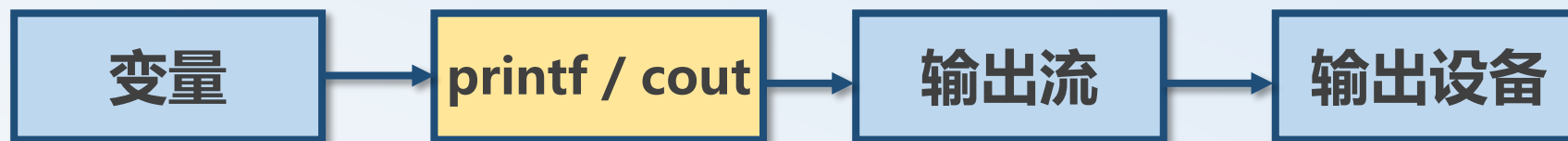
新的I/O

❖ C++语言的输入输出

➤ 输入过程



➤ 输出过程



新的I/O

```
#include <iostream>
using namespace std;
int main(void)
{
    int x;
    double y;
    /* 以下语句等价于printf("请输入两个整数(用逗号隔开):"); */
    cout << "请输入两个整数(用空格隔开) : ";
    /* 以下语句等价于scanf("%d %f", &x, &y); */
    cin >> x >> y;
    /* 以下语句等价于printf("x = %d, y = %f\n", x, y); */
    cout << "x = " << x << ", y = " << y << endl;
    return 0;
}
```

新的I/O

❖基本用法：

`cout << 表达式1 << 表达式2 << 表达式n;`

`cin >> 变量1 >> 变量2 >> 变量n;`

❖例如：

`cout << "x + y = " << x + y << "." << endl;`

`cin >> x >> y;`

新的I/O

❖ [注意]

- ① 不能用一个 << 输出多个数据项

```
cout<<a,b,c<<endl;
```

/* 错误 */

```
cout<<a<<b<<c<<endl;
```

/* 正确 */

- ② cin/cout可以分成多行来写

```
cin >> a >> b
```

```
>> c;
```

```
cout << a << b
```

```
<< c;
```

新的I/O - cout的用法举例

```
#include <iostream>
using namespace std;
```

```
int main(void)
{
```

```
    cout << "This is a C++ program! " << endl;
```

```
    cout << "This is"
         << " a C++"
         << "program!"
         << endl;
```

```
    return 0;
```

```
}
```

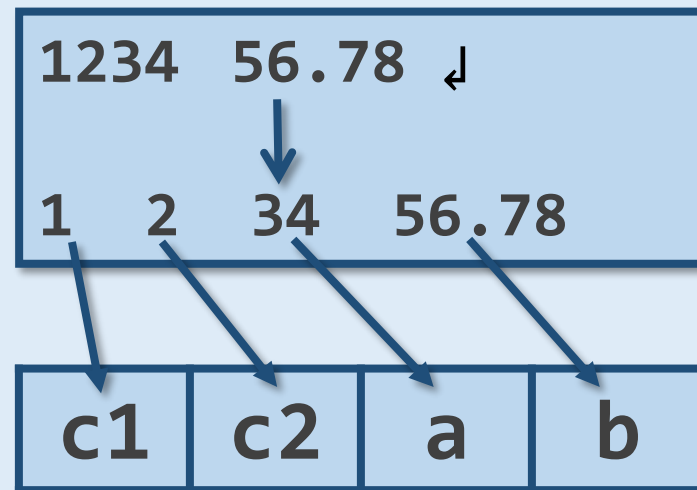
不分行

分多行

新的I/O - cin的用法举例

③ 不能用一个 << 输出多个数据项

```
int main(void)
{
    char  c1, c2;
    int  a;
    float b;
    cin >> c1 >> c2 >> a >> b;
    cout << "c1 = " << c1 << endl
         << "c2 = " << c2 << endl
         << "a = " << a << endl
         << "b = " << b << endl;
    return 0;
}
```



在I/O时, 系统会自动 根据
数据类型输入输出相应数据

新的I/O - cout与输出控制字符

❖想想为什么要有输出控制字符？

- 想用16进制形式输出
- 想控制输出宽度
- 想控制输出对齐方式....

❖回想printf的转换说明

❖C++有两种方法控制格式输出：

- 用格式控制符，必须包含头文`#include <iomanip>`
- 用流对象的成员函数(第7章讨论)

新的I/O – cout与输出控制字符

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(void)
{
    int x = 0;
    cout << "请输入一个八进制整数(以0开始) : ";
    cin >> oct >> x;
    cout << "x的十六进制表示为 : " << hex << x << endl;
    cout << "x的十进制表示为 : " << dec << x << endl;
    cout << "x的八进制表示为 : " << oct << x << endl;

    return 0;
}
```

输出控制字符

新的I/O – 输出控制字符

控 制 符	作 用
dec	设置基数为10
hex	设置基数为16
oct	设置基数为8
setbase(n)	设置整数的基数为n (n只能为8,10,16)
setfill(c)	设置填充字符c
setprecision(n)	设置显示实数精度为n位，以固定小数或指数输出时n为小数位数
setw(n)	字段宽度n位，不足n位左补空格
setiosflags(ios::fixed)	实数以固定小数显示
setiosflags(ios::scientific)	以指数形式显示

新的I/O – 输出控制字符

控 制 符	作 用
<code>setiosflags(ios::left)</code>	左对齐
<code>setiosflags(ios::right)</code>	右对齐
<code>setiosflags(ios::skipws)</code>	忽略前导空格
<code>setiosflags(ios::uppercase)</code>	16进制输出以大写字母显示A – F 科学计数法输出E
<code>setiosflags(ios::lowercase)</code>	16进制输出以小写字母显示a – f 科学计数法输出e
<code>setiosflags(ios::showpos)</code>	输出整数显示正号
<code>resetflags()</code>	终止已设置的输出格式状态，在括号中应该制定内容

[注意] 若用控制符，程序需包含头文件`#include <iomanip>`

1

C与C++的基本区别

新的初始化方法

新的I/O

新的注释

名字空间

有关类型的区别

2

独具魅力的const修饰符

新的注释 - 单行注释

❖ 单行注释

- **//** 可以嵌套 **//** 或 **/*... */**

例：int x; **// x是一个整型变量 // int是整型类型**

例：int x; **// x是一个整型变量 /* int是整型类型 */**

- **/*... */**方式的注释不能嵌套**/*... */**

- **/*... */**方式下可以嵌套**//**注释

例：int x; **/* x是一个整型变量 // int是整型类型 */**

新的注释 - 多行注释

```
#include <iostream> /*cin,cout在该文件中声明,注意".h"没有*/  
using namespace std; //cin、cout所在的名字空间,今后将讲述
```

```
/*
```

功能：测试输出控制字符

```
*/
```

```
int main(void)
```

```
{
```

```
    int x;
```

```
    cout << "请输入一个八进制整数(以开始) : ";
```

```
    cin >> oct >> x;
```

```
    cout << "x的八进制表示为 : " << oct << x << endl;
```

```
    return 0;
```

```
}
```


1

C与C++的基本区别

新的初始化方法

新的I/O

新的注释

名字空间

有关类型的区别

2

独具魅力的const修饰符

名字空间

```
void foo1()  
{  
    int x;  
    // TODO  
}
```

命名空间1

```
void foo2()  
{  
    int x;  
    //TODO  
}
```

命名空间2

不同名字空间中允许有
相同名称的标识符

可见，**名字空间**的作用是为了**防止“名字”冲突**

名字空间

❖ 名字空间声明语法

```
namespace <名字空间名称>
{
    // 标识符等
}
```

[注意]

- 名字空间的名称要符合标识符命名规则
- 若省略名字空间名称则名字空间只能在本文件内使用

名字空间 - 使用举例

```
#include <iostream>
using namespace std; /*cin、cout所在的名字空间，今后将讲述*/
namespace ns1
{
    int x = 1;
}
namespace ns2
{
    int x = 2;
}
int main(void)
{
    cout << "名字空间ns1中x的值为: " << ns1::x << endl;
    cout << "名字空间ns2中x的值为: " << ns2::x << endl;
    return 0;
}
```

作用域运算符

名字空间 - 使用举例

❖ 使用名字空间中的“名字”

方法1 : 名字空间名::标识符

方法2 : using 名字空间名::标识符;

方法3 : using namespace 名字空间名称;

名字空间

```
#include <iostream>
int main(void)
{
    std::cout << "Hello World!" << std::endl;

    return 0;
}
```

这种方式有什么缺点？

这种方式有什么缺点？

```
#include <iostream>
using namespace std;
int main(void)
{
    cout << "Hello World!" << endl;

    return 0;
}
```

名字空间

```
#include <iostream>
using std::cout;
using std::endl;
```

推荐！用到什么using什么！

```
int main(void)
{
    cout << "Hello World!" << endl;

    return 0;
}
```

名字空间

```
#include <iostream>
using namespace std;

namespace one
{
    const int M = 200;
    int inf = 10;
}

namespace two
{
    int x;
    int inf = -100;
}
```

```
using namespace one;
int main(void)
{
    using two::x;
    x = -100;
    cout << inf << endl; // 10
    cout << M << endl;   // 200
    two::inf *= 2;
    cout << two::inf << endl; // -200
    cout << x << endl;      // -100

    return 0;
}
```


1

C与C++的基本区别

新的初始化方法

新的I/O

新的注释

名字空间

有关类型的区别

2

独具魅力的const修饰符

有关类型的区别 - bool类型

	逻辑类型	真	假
C	没提供	非0	0
C++	bool	true	false

[注意]

- ① bool类型的取值只有两种true,false
- ② 输出时默认输出0或者1
- ③ 用**boolalpha**可以改变默认的输出方式，**noboolalpha**可以恢复默认的输出方式

有关类型的区别 - bool类型

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

```
{
```

```
    bool bval1 = 1 < 2;
```

```
    bool bval2 = true;
```

```
    bool bval3 = false;
```

```
    bool bval4 = 4;
```

```
    bool bval5 = 0;
```

```
    cout << "bval1=" << bval1 << endl;
```

```
    cout << "boolalpha bval1=" << boolalpha << bval1 << endl;
```

```
    cout << "noboolalpha bval1=" << noboolalpha << bval1 << endl;
```

```
    cout << "bval2=" << bval2 << endl;
```

```
    cout << "bval3=" << bval3 << endl;
```

```
    cout << "bval4=" << bval4 << endl;
```

```
    cout << "bval5=" << bval5 << endl;
```

```
    return 0;
```

```
}
```

均将隐式转换

```
bval1=1
boolalpha bval1=true
noboolalpha bval1=1
bval2=1
bval3=0
bval4=1
bval5=0
```

boolalpha
noboolalpha
改变输出方式

有关类型的区别 - string类

```
#include <iostream>
#include <string>
using namespace std;
```

必须包含相关头文件！

```
int main(void)
{
    string name = "student";
    string address = "Hebei... ..";
    cout << name << address << endl;
    return 0;
}
```

有关类型的区别 - string类

❖ string对象的定义和初始化

初始化string对象的方式	
<code>string s1;</code>	默认构造函数，s1为空串
<code>string s2(s1);</code>	将s2初始化为s1的一个副本
<code>string s3("value");</code>	用字符串字面值初始化s3
<code>String s4(n, 'c')</code>	将s4初始化为字符'c'的n个副本

有关类型的区别 - string类

```
#include <iostream>
#include <string>
using namespace std;
string s0;
int main(void)
{
    string s1;
    // string s2 = "hello world!";
    string s2("hello world!");
    // string s3 = s2;
    string s3(s2);
    string s4(5, 'r');
    cout << "s0=" <<s0 <<endl;
    cout << "s1=" <<s1 <<endl;
    cout << "s2=" <<s2 <<endl;
    cout << "s3=" <<s3 <<endl;
    cout << "s4=" <<s4 <<endl;
    return 0;
}
```

初始化string对象

有关类型的区别 - string类

❖ string对象的读写：用cin、cout读写string对象

[注意]

- cin忽略开头所有空格、TAB、回车符
- 不接收含空格的字符串

有关类型的区别 - string类

```
#include <iostream>
#include <string>
using namespace std;

int main(void)
{
    string s;
    cin >> s; //hello world!
    cout << s << endl; //hello

    return 0;
}
```

初始化string对象

```
#include <iostream>
#include <string>
using namespace std;

int main(void)
{
    string word;
    while(cin >> word)
        cout << word << endl;

    return 0;
}
```

string读入未知数目的对象

有关类型的区别 - string类

❖ string对象的操作，设有：string s, s1;

string的操做	
s.empty()	若s为空串，则返回true，否则返回false
s.size()	返回s中字符的个数
s[n]	返回s中位置为n的字符，位置从0开始
s1 + s2	将两个串连接成新串，返回新生成的串
s1 = s2	把s1得内容替换为s2的副本
v1 == v2	判定时候相等，相等返回true，否则返回false
!= < <= > >=	保持这些操作惯有的含义，如：s != s2;

操作方法小结: **string变量名.成员函数名()**

有关类型的区别 - string类

[注意]

- ① `size()`的返回类型并非`int`而是`string::size_type`类型的值。

建议不要把`size()`的返回值赋值给`int`变量。

```
string s2 = "hello";  
string::size_type count = s2.size();
```

有关类型的区别 - string类

- ② 两个string对象+时，+操作符左右操作数必须至少有一个是string

```
string s1 = "hello"  
string s2 = "world";  
string s3 = s1 + ",";  
string s4 = "hello" + "world "; ❌  
string s5 = "hello" + s2 + "world" ;
```

有关类型的区别 - string类

- ③ string对象下标操作时，任何无符号整型值均可用作下标，但下标的实际类型为`string::size_type`
- ④ string下标操作可用作左值

```
int main(void)
{
    string str = "student";
    cout << str << endl;
    for(string::size_type ix = 0; ix!=str.size(); ++ix)
        str[ix] = 'x';
    cout << str << endl;
    return 0;
}
```

有关类型的区别 - 枚举

❖ 回忆枚举

```
enum weekday  
{sun, mon, tue, wed, thu, fri, sat};
```

```
enum weekday workday, week_end;
```

```
enum weekday  
{sun, mon, tue, wed, thu, fri, sat} workday;
```

有关类型的区别 - 枚举

❖ C++对枚举的改进

- ① 在C++中定义枚举变量可以不用enum

```
enum weekday  
{sun, mon, tue, wed, thu, fri, sat};  
weekday w; // 省略了enum
```

- ② 无名枚举：不给出枚举类型名和变量，将枚举元素当符号常量用

```
enum  
{min = 0, max = 100};  
int x = min, arr[max];  
... ..
```

有关类型的区别 - 共用体

❖ 回忆共用体（联合体）

```
union 共用体名
{
    成员
};
union 共用体名 变量表
```

为最长成员的长度,可以赋初值,可以做返回值和参数,类型相同的可以相互赋值

```
union [共用体名]
{
    成员
}变量表;
```

有关类型的区别 - 共用体

❖ C++对联合的扩展

① 无名联合:没有联合体类型名和变量名的联合体

```
#include <iostream>
using namespace std;
int main(void)
{
    union {
        char c;
        int i;
        double d;
    } A;
    A.c = 'a';
    cout << c << endl;
    return 0;
}
```

特点：可直接引用数据项

有关类型的区别 - 共用体

② 定义联合变量无需给出union

```
#include <iostream>
using namespace std;
union test
{
    char c;
    int i;
    double d;
};
int main(void)
{
    test m = {'a'};
    cout << m.c << endl;
    return 0;
}
```

有关类型的区别 - 结构体

❖ C++对结构体的扩展

① 定义结构体变量可以不用struct

```
struct point
{
    double x;
    int a;
};
point p;
```

② 成员可以包含函数定义

```
struct point{
    double x,y;    //数据成员
    void setvalue(double a,double b) //成员函数
    {
        x = a;
        y = b;
    }
};
```

注：其他区别
第三章介绍

1

C与C++的基本区别

新的初始化方法

新的I/O

新的注释

名字空间

有关类型的区别

2

独具魅力的const修饰符

const

```
int y;  
int * py = &y;
```

```
const int x = 3;  
const int * px = &x;
```

```
int * const py2 = &y;  
const int * const px2 = &x;
```

```
int ** p1;  
const int ** p2 = ??; //应如何赋值?  
const int * const * p3 = ??; //应如何赋值?  
const int * const * const p4 = ??; //应如何赋值?
```

const - 变量与常量

`int x = 3;` // 变量

变量名

存储地址

存储内容

x

&x

3

`const int x = 3;` // 常量(只读变量)

变量名

存储地址

存储内容

x

&x

3

const - 变量与常量

❖ [注意]

- 只读变量必须初始化，且一旦定义则不能改变
`const double Kdval = 5.2;` // 必须初始化！
- 为了与C中的常量区别，我们将C中的常量称为**字面值常量**
3, 3.4, "hello world!"
- 如省略类型默认为int (**不推荐定义是省略类型**)
`const Kival = 4;` // 默认类型为int, 不推荐
- 常量首写字母大K开头且首字母大写，其余小写

const - 变量与常量

- 推荐使用常量（只读变量），尽量不用宏

```
#define T1 x+x
#define T2 T1-T1
int main(void)
{
    int x = 5;
    cout << "T1=" << T1
        << endl;
    cout << "T2=" << T2
        << endl;
    return 0;
}
```

```
int main(void)
{
    int x = 5;
    const int T1 = x+x;
    const int T2 = T1-T1;

    cout << "T1= " << T1 << endl;
    cout << "T2=" << T2 << endl;
    return 0;
}
```

const - 变量与常量

- const只读变量默认为文件局部变量,通过制定const变量为extern,就可以在整个程序中访问const对象

```
// FILE1
```

```
extern const int counter = 100;
```

```
// FILE2
```

```
#include <iostream>
```

```
using namespace std;
```

```
extern const int counter; // 声明counter为外部变量
```

```
int main(void)
```

```
{
```

```
    cout << counter << endl;
```

```
    return 0;
```

```
}
```


const - 一重常指针

```
int x = 3; int * const p = &x;  
p=&y; // 错误
```

变量名

x

p

存储地址

&x

&p

存储内容

3

&x

*p可变, p不可变, x可变

const - 一重常指针

```
int x = 3;          const int * p = &x;  
p = &y; // 正确  
*p = 4; // 错误
```

变量名

x

p

存储地址

&x

&p

存储内容

3

&x

*p不可变，p可变，x可变

const - 一重常指针

注意：

➤ `const int * to int * is Error`

在指针的赋值过程中，要注意保证对于指向变量的操作权限不可放大。

const - 一重常指针

```
int main(void)
{
    int x = 5;
    const int *p = NULL; // p可变, *p不可变
    p = &x;
    cout << "x=" << *p << endl;    //5
    cout << "++x=" << ++x << endl; //6
    // cout << (*p)++ << endl;
    int y(6);
    p = &y;
    cout << "*p=" << *p << endl;    //6
    return 0;
}
```

const - 多重常指针

```
const int x = 3; const int * const p = &x;  
p = &y; // 错误  
*p = 4; // 错误
```

变量名

x

p

存储地址

&x

&p

存储内容

3

&x

*p不可变, p不可变, x不可变

const - 一重常指针

```
int main(void)
{
    int x = 5;
    const int *const p = &x;
    // *p = 60;
    cout<<"*p="<<*p<<endl;
    int y = 6;
    // p = &y;

    return 0;
}
```

const - 二重常指针

```
int x = 3, y = 4;  
int * p1 = &x;  
int * const * p2 = &p1;  
*p2 = &y; // 错误  
p1 = &y; // 正确
```

p2 , **p2可变 , *p2不可变

变量名	变量地址	存储单元中的内容
p2	&p2	&p1
p1	&p1	&x
x	&x	3

const - 二重常指针

```
int x = 3;  
int *p1 = &x;  
int ** const p2 = &p1;  
**p2 = 5; // 正确  
x = 4;    // 正确
```

p2不可变, **p2可变,
*p2可变

变量名	变量地址	存储单元中的内容
p2	&p2	&p1
p1	&p1	&x
x	&x	3

const - 二重常指针

```
int x = 3; int * p1 = &x, * p0 = &x;  
int * const * const p2 = &p1;  
p2 = &p0; // 错误  
*p2 = &y; // 错误  
x = 4; // 正确
```

p2 , *p2不可变,
**p2可变

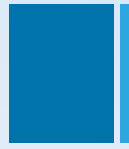
变量名	变量地址	存储单元中的内容
p2	&p2	&p1
p1	&p1	&x
x	&x	3

const - 二重常指针

```
int x = 3;    int * p1 = &x;  
const int * const * const p2 = &p1;  
p2 = &p0;    // 错误  
*p2 = &y;    // 错误  
**p2 = 5;    // 错误  
x = 4;       // 正确
```

p2、**p2、
*p2均不可变

变量名	变量地址	存储单元中的内容
p2	&p2	&p1
p1	&p1	&x
x	&x	3



const - 二重常指针

紧跟const的表达式不可变！其他可变。

本讲教学目标

- 理解赋值和初始化的区别
- 了解C++中的注释的写法
- 了解C++中的输入输出的写法
- 理解命名空间的概念
- 了解C++中的数据类型
- 掌握C++中const的使用



THANKS

