程序员的C++速成

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

内容概要

- C++的优势
- 程序结构
- 数据类型
- 变量定义和存储
- 输入输出
- 指针
- 运算符重载
- 引用类型
- 泛型程序设计

C++语言的优势

C++语言的优势

- 编译型语言，程序执行速度快(比如适合大型游戏开发)
- 可以做贴近硬件的底层编程
- C++可以做Java和Python做不了的事，C++可以作任何事
- 学习C++可以加深对计算机系统的理解

程序结构

内蒙古阿斯哈图石林

程序结构

C++程序从main函数开始执行。操作系统调用main函数:

main.cpp

```cpp
#include <iostream>      //包含头文件，类似import

using namespace std;     //使用std名字空间，暂且照抄不解释

int main()  //函数返回值类型为int
{
    cout << "Hello,world!" << endl;    //输出语句，endl表示换行
    return 0;
}
```

```
/*
如果不考虑美观，
从语法规则来说程序缩进随意。
*/
```

数据类型

瑞士布里茨恩湖

数据类型

bool	取值 true或false
int	32位的整数
long	32位的整数
long long	64位的整数
char	字符，一个字节
float	小数（浮点数），32位
double	小数(浮点数),64 位
unsigned int	无符号整数，总是被当作非负数
unsigned char	无符号字符
unsigned long	
....	

变量定义和存储

法国圣十字湖

变量定义

- 变量必须先定义后使用。定义变量时必须指定类型，且变量一旦定义，类型不可变

```
5  int main()
6  {
7      int a;
8      a = 13;
9      string s = "hello";
10     a = "world";                    //类型不匹配导致编译错误
11 }
```

变量存储方式（重要!!!重要!!!重要!!!)

- Python所有变量都相当于C++的指针，指向数据
- Java除基本类型以外的变量都相当于C++的指针，指向数据
- C++的变量本身就存储数据，类似于Java的基本类型(int,float.....)变量

Python中的变量都是指针

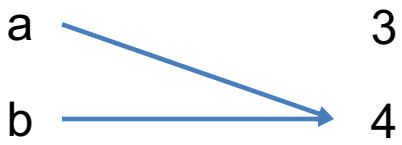
- 对变量进行赋值，意味着将变量指向某处

```
a = 3
b = 4
```

Python中的变量都是指针

- 用一个变量对另一个变量赋值意味着让两个变量指向同一个地方

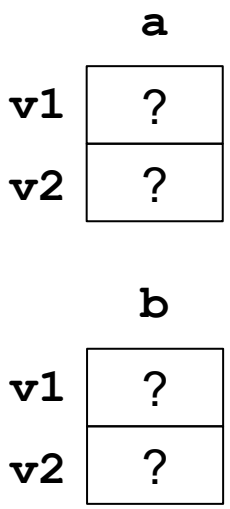
```
a = b
```



Java中的对象和Python变量一样，都是指针

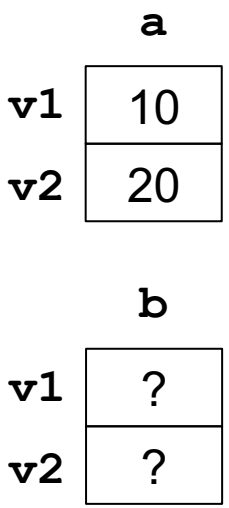
C++变量自带存储空间

```
class A {   //类A
  public:
    int v1,v2;   //两个成员变量
    int get() {  //成员函数
        return v1;
    }
};
int main() {
    cout << sizeof(A) << endl; //输出8,A对象占字节数
    A a,b;   //a,b立即各占8字节存储空间
```



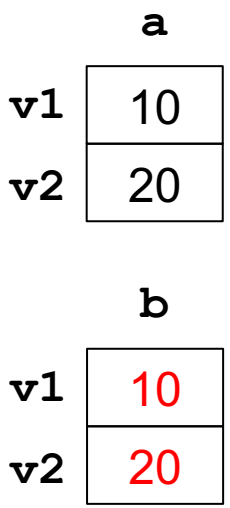
C++变量自带存储空间

```
a.v1 = 10;  a.v2 = 20;
```



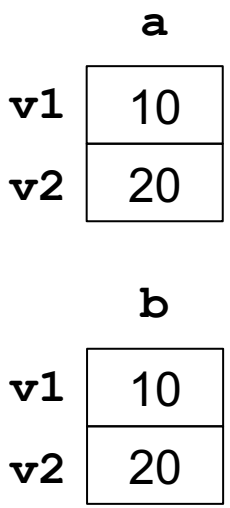
C++变量自带存储空间

```
a.v1 = 10;  a.v2 = 20;
b = a; //将a内容复制到b
```



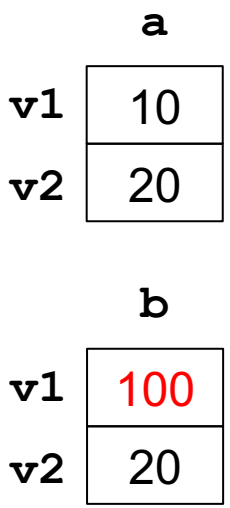
C++变量自带存储空间

```
a.v1 = 10;  a.v2 = 20;
b = a; //将a内容复制到b
cout << b.v1 << endl;    //输出10
```



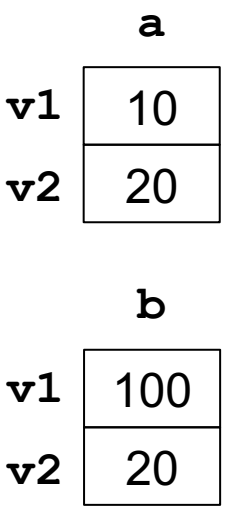
C++变量自带存储空间

```
a.v1 = 10;  a.v2 = 20;
b = a; //将a内容复制到b
cout << b.v1 << endl;    //输出10
b.v1 = 100;  //不会影响到a
cout << a.get() << endl;  //输出10
```



C++变量自带存储空间

```
    a.v1 = 10;  a.v2 = 20;
    b = a; //将a内容复制到b
    cout << b.v1 << endl;    //输出10
    b.v1 = 100;  //不会影响到a
    cout << a.get() << endl;  //输出10
    a == b;  //编译错。默认情况下自定义类的对象不可以比较
    return 0;
}
```



输入输出

输入输出

- cin用于输入，cout用于输出

main.cpp

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    int a, b;
    string s1, s2;
    cin >> a >> b >> s1 >> s2;
    //输入,各项之间用空格或换行分隔
    cout << a + b << "," << s1 + s2 << endl;
    return 0;
}
```

sample0.cpp

输入:
3 5 Hello World
输出:
8,HelloWorld

指针

德国国王湖

指针

- 尽管Python变量和Java对象本质上就是指针，但是很少有语言和C/C++一样有指针概念

- 指针中存放的，是变量（或常量）的内存地址

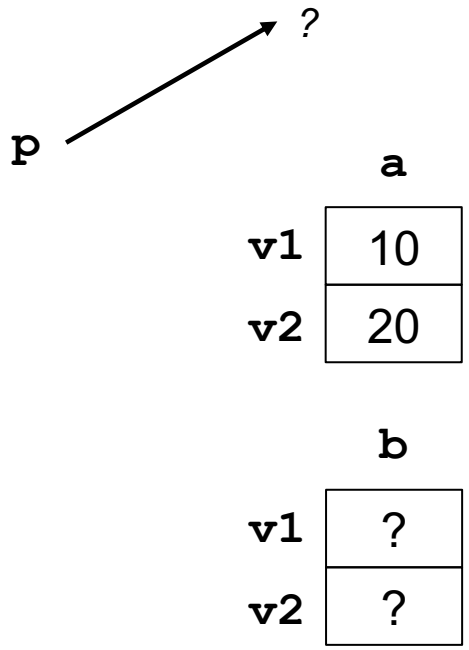
- 指针的意义：提供随意访问内存的手段。不需要支持贴近硬件的底层程序设计的语言，不需要支持指针。

- 定义指针： `T * p;` T是类型。

  如 `int * p, char * p; float *p`

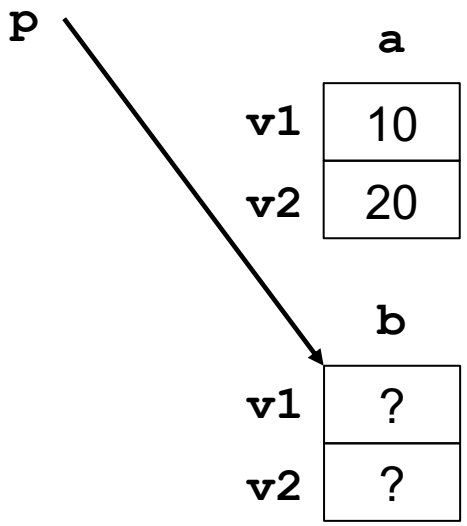
指针

```
class A {   //类A
public:
    int v1,v2;   //两个成员变量
};
int main() {
    A * p;   //p是一个指针，用来指向A类型的变量
    A a,b;
    a.v1 = 10;  a.v2 = 20;
```



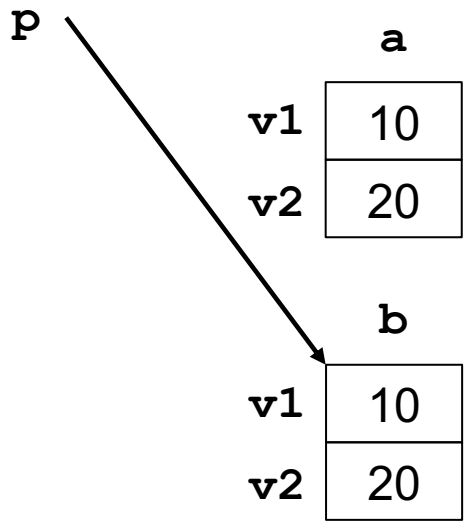
指针

```
class A {   //类A
public:
    int v1,v2;   //两个成员变量
};
int main() {
    A * p;   //p是一个指针，用来指向A类型的变量
    A a,b;
    a.v1 = 10;  a.v2 = 20;
    p = & b; //让p指向b, &b 是取b的地址
```



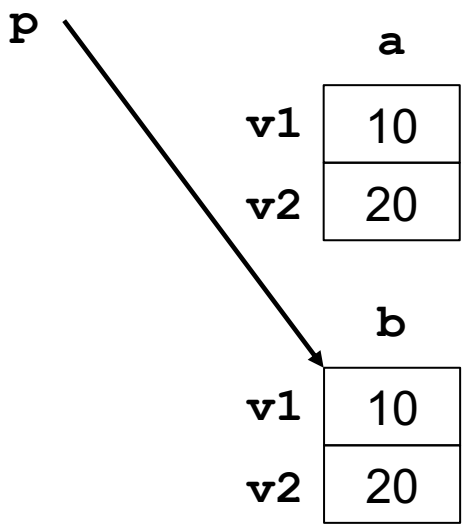
指针

```
class A {   //类A
public:
    int v1,v2;   //两个成员变量
};
int main() {
    A * p;   //p是一个指针，用来指向A类型的变量
    A a,b;
    a.v1 = 10;  a.v2 = 20;
    p = & b; //让p指向b, &b 是取a的地址
    * p = a; // * p表示p指向的变量，本句等价于 b = a;
```



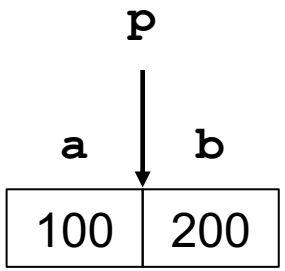
指针

```
    cout << b.v1 << endl; //输出10
    cout << p->v2 << endl;
    //输出20 p->v2表示p所指向的变量的v2
    return 0;
}
```



指针提供随意访问内存的能力

```
int a = 100, b = 200;
int main() {
    int * p = & b;   //p指向b
```

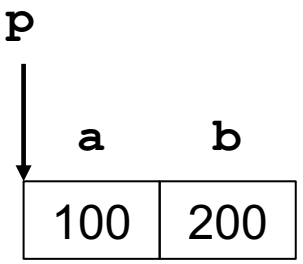


Clion 2022.2编译出来的程序中，变量a和变量b在内存挨着存放

t2.cpp

指针提供随意访问内存的能力

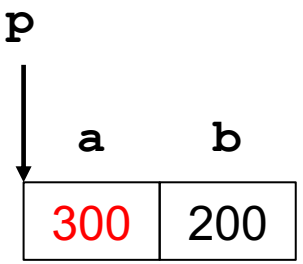
```
int a = 100, b = 200;
int main() {
    int * p = & b;   //p指向b
    p = p - 1;       //使p指向b的地址前面的4个字节
```



Clion 2022.2编译出来的程序中，变量a和变量b在内存挨着存放

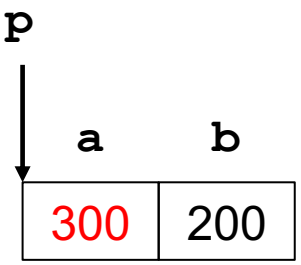
指针提供随意访问内存的能力

```
int a = 100, b = 200;
int main() {
    int * p = & b;   //p指向b
    p = p - 1;       //p指向b的地址前面的4个字节
    * p = 300;       //往p指向的地方写入300
```



指针提供随意访问内存的能力

```
int a = 100, b = 200;
int main() {
    int * p = & b;   //p指向b
    p = p - 1;       //p指向b的地址前面的4个字节
    * p = 300;       //往p指向的地方写入300
    cout << a << endl;  //输出300
    cout << hex << &a << "," << &b << "," << p << endl;
    //输出a的地址、b的地址、p中存放的地址
    //输出: 0x7ff6fcbc4010,0x7ff6fcbc4014,0x7ff6fcbc4010
    return 0;
} //Clion 2022.2编译的程序运行结果如上，换个编译器未必是这个结果
```



数组和指针的关系

- 数组的名字就是值不可修改的指针

```
#include <iostream>
using namespace std;
int main()  {
    int * p;
    int a[] = {1,2,3,4,5,6};
    p = a; //数组名字是指针，代表数组的起始地址。
    for(int i = 0;i < 6;++i) { //>>1,2,3,4,5,6,
        cout << * p << ",";
        p += 1;  //p指向a的下一个元素
    }
```

数组和指针的关系

- 数组的名字就是值不可修改的指针

```
    a = a + 1; //编译错，数组名字的值不可修改
    a = p; //编译错
    return 0;
}
```

C++倾向于相信程序员的能力并少做限制

- C++数组越界的程序可能依然可以运行

sample3.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main() {  //在CLion 2022.2编译运行结果如下，换编译器结果不确定
6       int a = 0;
7       int b[] = {[0]:1, [1]:2, [2]:3, [3]:4};
8       b[-10] = 200;  //明显数组越界，但运行到此未必出错
9       for (int i = 0; i < 15; ++i)
10          b[i] = 200; //明显数组越界，但运行到此未必出错
11      cout << a << endl;  //输出200，a存放在b后面，其值因上条语句而改变
12      cout << hex << &a << "," << b << endl;
13      //数组名字是指针，代表数组的起始地址。输出 0xe0bdfff608,0xe0bdfff5f0
14      return 0;
15  }
```

字符串和指针

- C++ 有一种字符串就是字符数组

```
#include <iostream>
using namespace std;
int main()  {
    const char * p = "Hello";
    char a[] = "World";
    cout << p << endl;  //输出Hello
    p = a;
    cout << p << endl;  //输出World
    p = a + 2;  //等价于 p = &a[2];
    cout << p << "," << p[0] << endl; //输出rld,r
    return 0;
}
```

运算符重载

美国拱门国家公园

运算符重载

- 可以解决自定义对象不能用 <,<=, ==, != ,>, >= 比较的问题

```
class A {
public:
    int n;
    A(int n_):n(n_) { }
    bool operator < (A b) {
        //重载 < 号
        return n < b.n;
    }
    bool operator == (A b) {
        return n == b.n;
    }
};
```

```
int main()  {
    A a1(4),a2(5);
    cout << (a1 == a2) << endl;
    //输出0 等价于 a1.operator==(a2);
    cout << (a1 < a2) << endl;
    //输出1 等价于a1.operator<(a2);
    return 0;
}
```

运算符重载

- 使得+,-,*,/等运算符可以用于自定义对象

```
class Int{
public:
    int n;
    Int(int n_):n(n_) { }
    Int operator + (Int b) {
        return Int(n + b.n);
    }
};
Int operator + (int i, Int a) {
    return Int(i + a.n);
}
```

```
int main()  {
    Int a1(4),a2(5);
    cout << (a1 + a2).n << endl;
    //输出9 等价于 a1.operator+(a2);
    cout << (3 + a1).n << endl;
    //输出7 等价于operator+(3,a1);
    return 0;
}
```

引用类型

荷兰阿姆斯特丹库肯霍夫公园

引用类型

- 一个变量的引用，和这个变量是一回事

定义引用的方式：

```
T & 变量名1 = 变量名2;
```

T是类型，变量名1引用了变量名2，从此两个变量是一回事

引用类型

```
#include <iostream>
using namespace std;
int main()  {
    int a = 5, b = 4;
    int & r = a; //r引用a以后就永远引用a，和a是一回事
    r = 200;
    cout << r << "," << a << endl; //>>200,200
    a = 500;
    cout << r << endl;  //>>500
    r = b;  //相当于对a赋值，不是让r引用b
    cout  << r << "," << a << endl; //>>4,4
    return 0;
}
```

函数参数传值

```
void Swap(int a,int b) {
//因参数传值(形参是实参的复制品)，不会导致实参变化
    int tmp = a;
    a = b;
    b = tmp;
}
int main()  {
    int a = 5, b = 4;
    Swap(a,b);
    cout  << a << "," << b << endl; //>>5,4
    return 0;
}
```

函数参数传引用

```
void Swap(int & a,int & b) {
//参数传引用，形参是实参的引用，和实参是一回事，会导致实参变化
    int tmp = a;
    a = b;
    b = tmp;
}
int main()  {
    int a = 5, b = 4;
    Swap(a,b);
    cout  << a << "," << b << endl; //>>4,5
    return 0;
}
```

泛型程序设计

美国鹅颈湾

泛型程序设计

- 所谓“泛型”，指的就是算法或数据结构只要实现一遍，就能适用于多种数据类型。具体做法是

- 编写一段代码时，其中的一些变量，类型并非具体确定的类型，而是用一个“类型参数”(例如T、E、Some等符合变量命名规则的标识符)来表示。当这段代码被真正应用的时候，“类型参数”可以被替换为不同的具体类型，如int,String,Student等。

- 例如，希望实现一个可变长的数组，用一套代码就可以生成int类型、String类型等不同类型的数组，就可以用泛型机制来实现。

泛型程序设计

- C++通过“模板”实现泛型。模板有函数模板和类模板
- Java支持“伪泛型”
- Python 变量类型是运行时确定，所以天然支持泛型

函数模板

```
#include <iostream>
#include <string>
using namespace std;
template <class T>  //函数模板
void Swap(T & a, T & b) {
    T tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

函数模板

```
int main()  {
    int a = 5, b = 4;
    Swap(a,b);
    cout << a << "," << b << endl; //>>4,5
    string s1 = "Hello", s2 = "World";
    Swap(s1,s2);
    cout << s1 << "," << s2 << endl; //>>World,Hello
    return 0;
}
```

```
void Swap(int & a, int & b) {
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

```
void Swap(string & a, string & b) {
    string tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

类模板

```
template <class T1,class T2> //类模板
class Pair {
public:
    T1 key;  //成员变量key是T1类型
    T2 value;  //成员变量value是T2类型
    Pair(T1 k,T2 v) {  //构造函数
        key = k; value = v;
    }
    T1 getKey() { //返回值是T1类型
        return key;
    }
};
```

类模板

```
int main() {
    Pair<int,string> a(100,"Tom");
 //Pair模板中T1替换成int,T2替换成string得类 Pair<int,string>
    int aKey = a.getKey();  //aKey值为100
    Pair<string,double> b("Jack",3.14);
 //Pair模板中T1替换成string,T2替换成double得类 Pair<string,double>
    string bKey = b.getKey();   //bKey值为 "Jack"
    return 0;
}
```

标准模板库STL

将一些常用的数据结构（比如链表，数组，二叉树）和算法（比如排序，查找）写成模板，以后则不论数据结构里放的是什么对象，算法针对什么样的对象，则都不必重新实现数据结构，重新编写算法。

标准模板库 (Standard Template Library) 就是一些常用数据结构和算法的模板的集合。

有了STL，不必再写大多的标准数据结构和算法，并且可获得非常高的性能。