



华中科技大学

# 面向对象程序设计

## 第二章 C 基础

许向阳

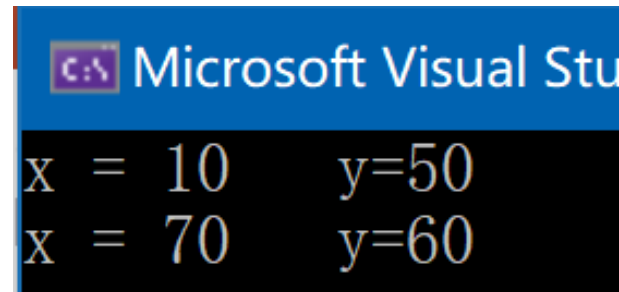
**xuxy@hust.edu.cn**





# 小测验

```
#include <iostream>
int main()
{
    int x=10;
    int a[3] = { 20,30,40 };
    int y = 50;
    printf("x = %d  y=%d\n", x, y);
    a[-1] = 60;
    a[3] = 70;
    printf("x = %d  y=%d\n", x, y);
    return 0;
}
```



C:\ Microsoft Visual Studio

x = 10	y=50
x = 70	y=60

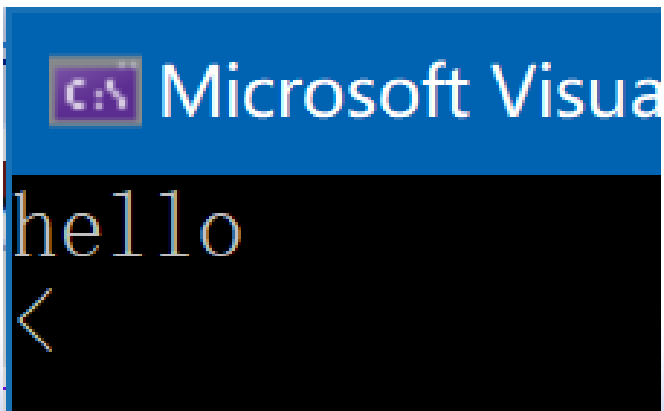
**Q: 运行结果是什么？为什么？**





# 小测验

```
#include <iostream>
char* f()
{
    char temp[20];
    strcpy_s(temp, "hello");
    return temp;
}
```



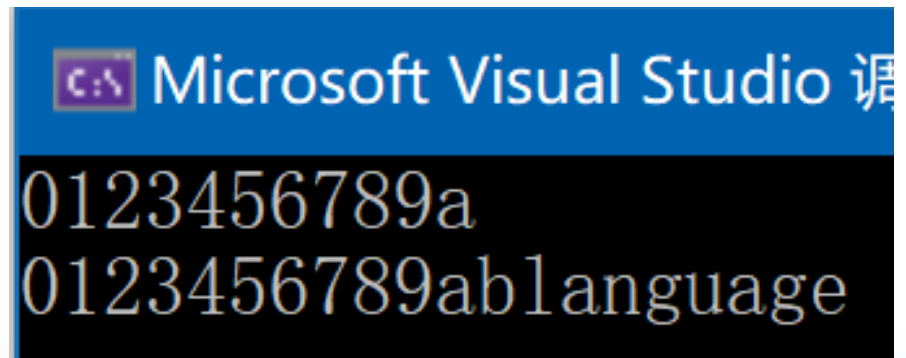
```
int main()
{
    char* p;
    char a[20];
    int i=0;
    p = f();
    while (*(p + i) != 0) {
        a[i] = p[i];
        i++;
    }
    a[i] = 0;
    printf("%s \n", a);
    printf("%s \n", p);
    return 0;
}
```



# 小测验

字符串 以 0为结束符

```
#include <iostream>
int main()
{
    char s1[20];
    char s2[12];
    strcpy_s(s2, "0123456789a");
    printf("%s\n", s2);
    s2[11] = 'b';
    strcpy_s(s1, "language");
    printf("%s\n", s2);
    return 0;
}
```





华中科技大学

# 学习内容

变量的三个属性

指针变量

地址类型转换

数据类型转换





# 变量的三个属性

- 数据类型
- 单元内容
- 单元地址

程序：对数据进行加工和处理

数据在哪里？ 数据的地址

数据地址有哪些表达方式？

地址表达的多样性





# 变量的三个属性

```
int x;  
x=10;
```

- 数据类型
- 单元地址
- 单元内容

0x005efb20  
&x

X=10

获取一个单元  
的地址 &x

```
int x;  
x = 10;  
printf("%x %x\n", x, (unsigned int)&x);  
return 0;  
}
```

68 % 未找到相关问题

监视 1

搜索(Ctrl+E) 搜索深度: 3

名称	值	类型
x	10	int
&x	0x005efb20 {10}	int *

C:\教学\本科 a 5efb20

对于简单类型的变量x，其地址为 &x

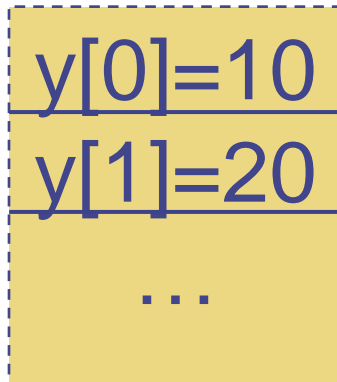




# 变量的三个属性

```
int y[5];  
int *p;  
p=y;  
p=&y[0];
```

y



&y[0] 0x0136f898  
&y[1] 0x0136f89c  
&y[2] 0x0136f8a0

```
int y[5];  
y[0] = 10;  
y[1] = 20;  
printf("%x %x %x\n", (unsigned int) y, (unsigned int)&y[0], unsigned int(&y[1]));  
return 0;
```

68 %

未找到相关问题

监视 1

搜索(Ctrl+E)



搜索深度: 3

名称	值	类型
▸ y	0x0136f898 {10, 20, -...	int[5]
▸ &y[0]	0x0136f898 {10}	int *
▸ &y[1]	0x0136f89c {20}	int *

C:\教学\本科教学\面向对象程序设计\面向对象程序设计例程\C语言基

136f898 136f898 136f89c

对于数组类型的变量y，其地址为 y





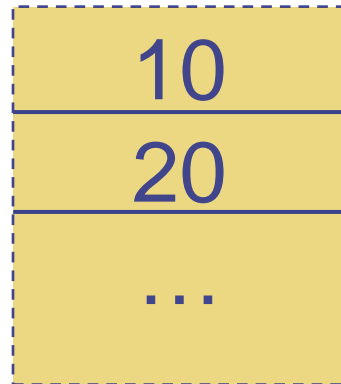
# 变量的三个属性

```
int y[5];
```

```
int x;
```

```
int *p;
```

y



y[0] 0x0136f898

y[1] 0x0136f89c

y[2] 0x0136f8a0

单元 x, y[0], y[1], y[2], .....

地址 &x, &y[0], &y[1], &y[2], .....

y[i] 的地址, 是 y 的地址 + i \* 元素的长度

```
p = y;
```

```
p = p + 1;
```

```
p = &y[0];
```

```
p = &y[1];
```

```
&p
```

```
&p
```

p = 0x136f898

p = 0x136f89c



# 变量的三个属性

```
int z[3][4];
```

```
int *p;
```

```
int *q[3];
```

q[0], q[1], q[2] 指向 int 类型的单元  
是数组， 每个元素都是指针， 指针数组

单元 z[0][0], z[0][1], z[0][2], .....

地址 &z[0][0], &z[0][1], &z[0][2], .....

```
p = &z[0][0];
```

```
p = p + 1;
```

```
q[0] = z[0];
```

```
q[0] = q[0] + 1;
```

```
p = z[0];
```

```
p = &z[0][1];
```

```
q[1] = z[1]; // 行的起始地址
```

```
q[0] = &z[0][1];
```

p=(int \*)z; 三者的功能等价





# 变量的三个属性

```
int z[3][4];
```

```
int *p;
```

```
int *q[3];
```

q[0], q[1], q[2] 指向 int 类型的单元  
是数组，每个元素都是指针，指针数组

```
int (*w)[4]; -> int u[4];  
            -> w -> u[4]
```

```
w = z;
```

```
w = (int (*)[4])&z[0][0];
```

```
w = (int (*)[4])z[0];
```

```
w = w + 1;      // w 指向 z 数组的第一行  
                // 一次增加数组一行的长度
```

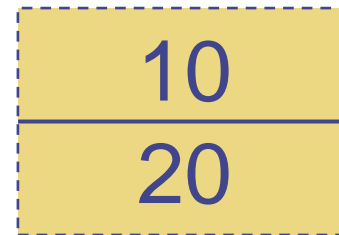







# 变量的三个属性

```
typedef struct {  
    int px;  
    int py;  
}point;
```

p1



```
point p1;  
point *p2;  
p1.px = 10;  
p1.py = 20;  
p2 = &p1;
```

名称	值	类型
▸  p1	{px=10 py=20 }	point
▸  &p1	0x00eff96c {px=10 py=20 }	point *
▸  p2	0x00eff96c {px=10 py=20 }	point *

对于结构类型的变量p1，其地址为 &p1





# 变量的三个属性

```
typedef struct {  
    int px;  
    int py;  
}point;
```

```
point p1[4];  
point *p2;
```

```
p2 = p1;  
p2 = &p1[0];
```



对于结构数组类型的变量p1，其地址为 p1



# 指针变量

```
int x;
```

```
x=10;
```

0x005efb20

x=10

```
*p=20;
```

```
*(&x)=20;
```

```
x=20;
```

```
int *p;
```

```
p=&x;
```

0x005efb34

p=

0x005ebf20

p

0x005efb34



# 指针变量

```
int x;
```

```
x=10;
```

```
int *p1;
```

```
p1=&x;
```

0x005efb20

x=10

0x005ebf20

p1

0x005efb34

```
**p2=20;
```

```
**(&p1)=20;
```

```
*p1=20;
```

```
*(&x)=20;
```

```
x=20;
```

```
int **p2;
```

```
p2=&p1;
```

0x005efb34

p2

&p2





# 指针变量

```
int x;  
x=10;
```

0x005efb20

x=10

```
**p2=20;
```

```
*(&(*(&x)))=20;
```

错误写法

```
**(&(&x))=20;
```

```
int *p1;  
p1=&x;
```

0x005ebf20  
p1

0x005efb34  
&p1

```
int **p2;  
p2=&p1;
```

0x005efb34  
p2

&p2







# 地址类型转换

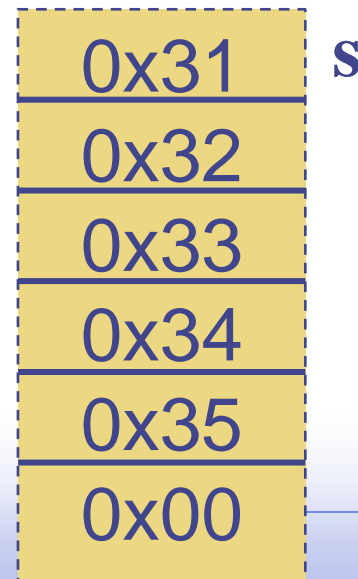
```
char s[10] = "12345";  
printf("%c %x\n", *s, *s);  
printf("%d %x\n", *(short *)s, *(short *)s);  
printf("%d %x\n", *(int *)s, *(int *)s);
```

**Q: 运行结果是什么？为什么？**

1 31

12849 3231

875770417 34333231





# 地址类型转换

```
union data {  
    char s[10];  
    int x;  
    short y;  
    char c;  
};
```

运行结果:

1 31

12849 3231

875770417 34333231






```
union data myunion;  
strcpy_s(myunion.s, "12345");  
printf("%c %x\n", myunion.c, myunion.c);  
printf("%d %x\n", myunion.y, myunion.y);  
printf("%d %x\n", myunion.x, myunion.x);
```


**Union 结构中，相同的内存单元，用不同类型解读**





# 地址类型转换

名称	值	类型
▸  &myunion.x	0x00affc14 {875770417}	int *
▸  &myunion.y	0x00affc14 {12849}	short *
▸  &myunion.c	0x00affc14 "12345" 	char *
▸  &myunion.s	0x00affc14 {49 '1', 50 '2', 51 '3', ...}	char[10] *

内存 1									
地址:	0x00AFFC14		列:	自动					
0x00AFFC14	31	32	33	34	35	00	fe	fe	12345.??
0x00AFFC1C	fe	fe	cc	cc	cc	cc	cc	cc	????????

相同的数值，因类型不同，表达的形式不同





# 数据类型转换

```
int x = 10;  
float y = 10;
```

**x=y;** 警告：从“float”转换到“int”，可能丢失数据

```
x = (int)y;          cvttss2si  eax,dword ptr [y]  
                    mov      dword ptr [x],eax
```

**y=x;** 警告：从“int”转换到“float”，可能丢失数据

```
y=float(x);
```





# 常见错误

数组越界

返回函数中局部变量的地址

指针指向的地址越界

字符串的复制





# 常见错误 数组越界

```
#include <iostream>
int main()
{
    int x=10;
    int a[3] = { 20,30,40 };
    int y = 50;
    printf("x = %d  y=%d\n", x, y);
    a[-1] = 60;
    a[3] = 70;
    printf("x = %d  y=%d\n", x, y);
    return 0;
}
```

**Q: 运行结果是什么？ 为什么？**



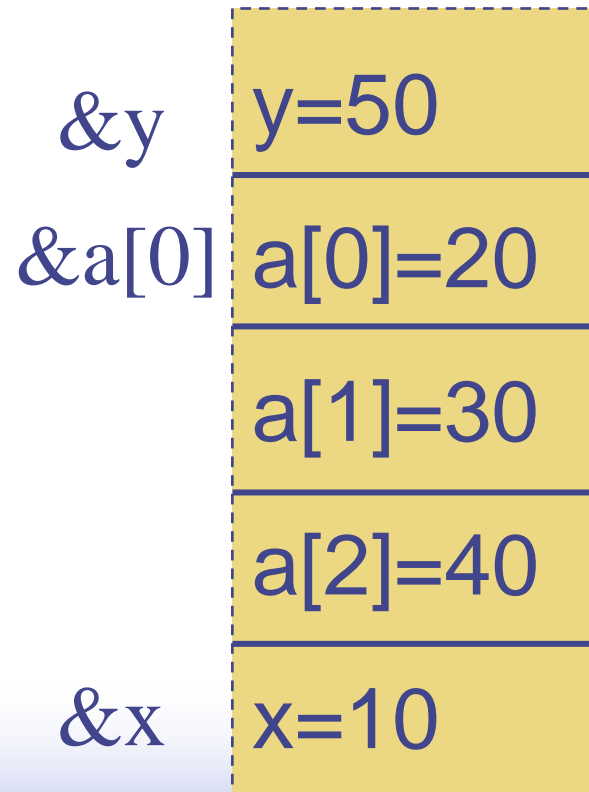


# 常見錯誤 数组越界

```
#include <iostream>
int main()
{
    int x=10;
    int a[3] = { 20,30,40 };
    int y = 50;
    printf("x = %d   y=%d\n", x, y);
    a[-1] = 60;
    a[3] = 70;
    printf("x = %d   y=%d\n", x, y);
    return 0;
}
```

C:\ Microsoft Visual Studio

```
x = 10    y=50
x = 70    y=60
```





# 常见错误 数组越界

配置(C): 活动(Debug) 平台(P): 活动(Win32)

配置属性	启用字符串池	
常规	启用最小重新生成	否 (/Gm-)
高级	启用 C++ 异常	是 (/EHsc)
调试	较小类型检查	否
VC++ 目录	基本运行时检查	默认值
C/C++	运行库	堆栈帧 (/RTCs)
常规	结构成员对齐	未初始化的变量 (/RTCu)
优化	安全检查	两者(/RTC1, 等同于 /RTCsu) (/RTC1)
预处理器	控制流防护	默认值
代码生成	启用函数级链接	<从父级或项目默认设置继承>
语言	启用并行代码生成	

项目属性: C/C++ -> 代码生成 -> 基本运行时检查 ,  
设置为默认值, 变量空间分配是紧凑的;  
才会出现上面的结果。







# 常见错误 数组越界

调试	较小类型检查	否
VC++ 目录	基本运行时检查	两者(/RTC1, 等同于 /RTCsu) (/RTC1)
▲ C/C++	运行库	多线程调试 DLL (/MDd)
常规	结构成员对齐	默认设置
优化	安全检查	启用安全检查 (/GS)
预处理器	控制流防护	
代码生成	启用函数级链接	

项目属性：C/C++ -> 代码生成 -> 基本运行时检查，  
设置为两者，  
变量空间分配非紧凑的，即变量不相邻，  
会出现异常对话框。



华中科技大学

# 常见错误 数组越界

C:\教学\本科教学\面向对象程序设计\面向对象程序设计例程\C语言基础\Debug\数组越界.exe

```
x = 10    y=50  
x = 10    y=50
```

Microsoft Visual C++ Runtime Library



Debug Error!

Program: C:\教学\本科教学\面向对象程序设计\面向对象程序设计例程\C语言基础\Debug\数组越界.exe

Module: C:\教学\本科教学\面向对象程序设计\面向对象程序设计例程\C语言基础\Debug\数组越界.exe

File:

Run-Time Check Failure #2 - Stack around the variable 'a' was corrupted.

(Press Retry to debug the application)

中止(A)

重试(R)

忽略(I)





# 常见错误 数组越界

## 思考题：

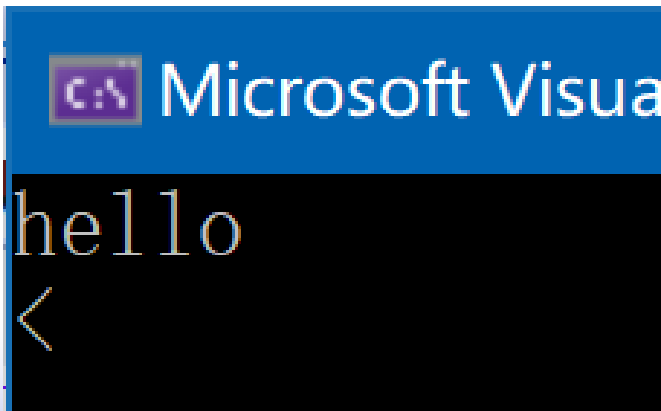
- 何时进行了越界检查？
- 如何进行越界检查？
- 能否发生了越界，而系统又未检测出来？
- 在没有越界检查时，数组越界是否不会导致程序崩溃？





# 常见错误 返回局部地址

```
#include <iostream>
char* f()
{
    char temp[20];
    strcpy_s(temp, "hello");
    return temp;
}
```



```
int main()
{
    char* p;
    char a[20];
    int i=0;
    p = f();
    while (*(p + i) != 0) {
        a[i] = p[i];
        i++;
    }
    a[i] = 0;
    printf("%s \n", a);
    printf("%s \n", p);
    return 0;
}
```





# 常见错误 返回局部地址

**warning C4172: 返回局部变量或临时变量的地址: temp**

```
a[i] = 0;  
printf("%s \n", a);  
printf("%s \n", p);  
return 0;
```

监视 1

搜索(Ctrl+E)



搜索深度: 3

名称

值

类型

▶



p

0x006ffb30 "hello"



char \*

```
a[i] = 0;  
printf("%s \n", a);  
printf("%s \n", p); 已用时间 <= 1ms  
return 0;
```

监视 1

搜索(Ctrl+E)



搜索深度: 3

名称

值

类型

▶



p

0x006ffb30 ""



char \*

执行printf("%s\n",a) 后,  
P 指向的单元未变  
但单元串中内容发生变化

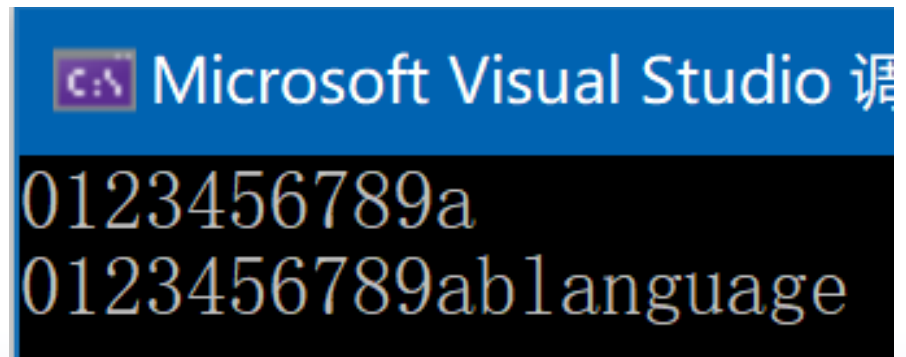




# 常见错误 字符串

字符串 以 0 为结束符

```
#include <iostream>
int main()
{
    char s1[20];
    char s2[12];
    strcpy_s(s2, "0123456789a");
    printf("%s\n", s2);
    s2[11] = 'b';
    strcpy_s(s1, "language");
    printf("%s\n", s2);
    return 0;
}
```

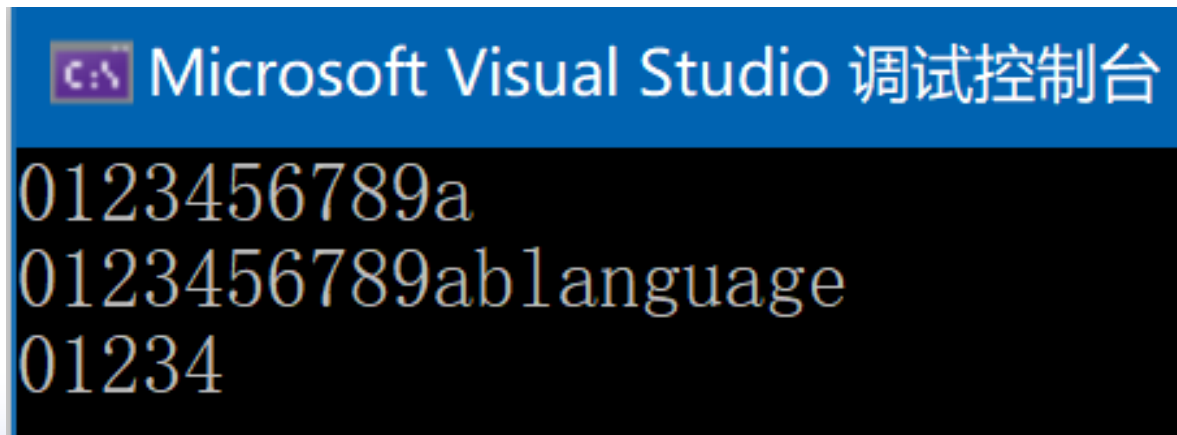




# 常见错误 字符串

字符串 以 0 为结束符

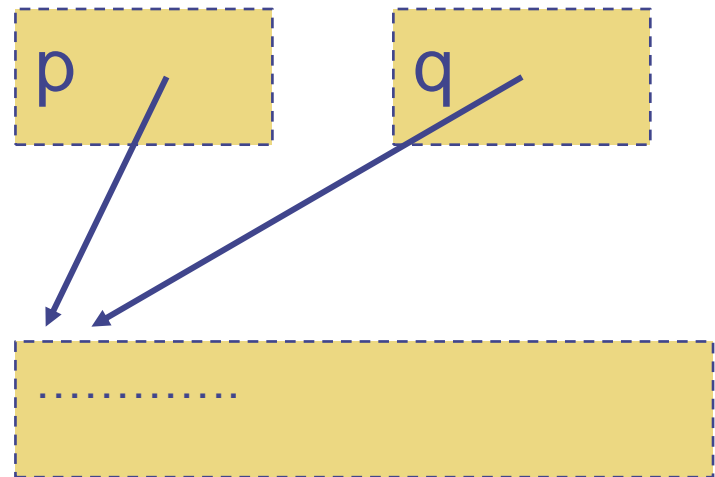
```
#include <iostream>
int main()
{ .....
    printf("%s\n", s2);
    s2[5] = 0;
    printf("%s\n", s2);
    return 0;
}
```



```
C:\> Microsoft Visual Studio 调试控制台
0123456789a
0123456789ablanguage
01234
```

# 常见错误 空间的分配与释放

```
#include <iostream>
int main()
{
    char* p, *q;
    p =(char *)malloc(100);
    q = p;
    if (p != NULL) {
        free(p);    p = NULL;
    }
    if (q != NULL) {
        free(q);    q = NULL;
    }
    return 0;
}
```



程序运行异常





# 常见错误 空间的分配与释放

```
#define CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#include <iostream>
int main()
{
    char* p;
    p = (char*)malloc(100);
    strcpy_s(p, 6, "hello");
    _CrtDumpMemoryLeaks();
    return 0;
}
```

```
Detected memory leaks!
Dumping objects ->
{76} normal block at 0x01435130, 100 bytes long.
Data: <hello          > 68 65 6C 6C 6F 00 CD CD CD CD CD CD CD CD CD
Object dump complete.
```





华中科技大学

