



华中科技大学

C 回顾与提高

许向阳

xuxy@hust.edu.cn





小测验

```
int a[10000][40000]; 将所有元素都置为0
```

```
for (i = 0; i < 10000; i++)           // 方法1
```

```
    for (j = 0; j < 40000; j++)
```

```
        a[i][j] = 0;
```

```
for (j = 0; j < 40000; j++)           // 方法2
```

```
    for (i = 0; i < 10000; i++)
```

```
        a[i][j] = 0;
```

```
memset(a, 0, sizeof(int)*10000 * 40000); // 方法3
```

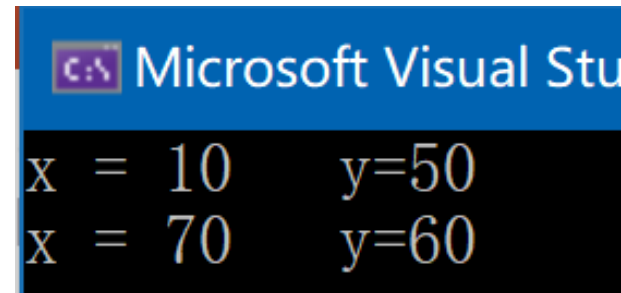
Q: 哪一种方法的运行速度快? 为什么?





小测验—数组访问

```
#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;
}
```



50	y
20	a[0]
30	a[1]
40	a[2]
10	x

Q: 运行结果是什么？ 为什么？





常见错误 数组越界

配置(C): 活动(Debug) 平台(P): 活动(Win32)

配置属性

常规

高级

调试

VC++ 目录

C/C++

常规

优化

预处理器

代码生成

语言

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

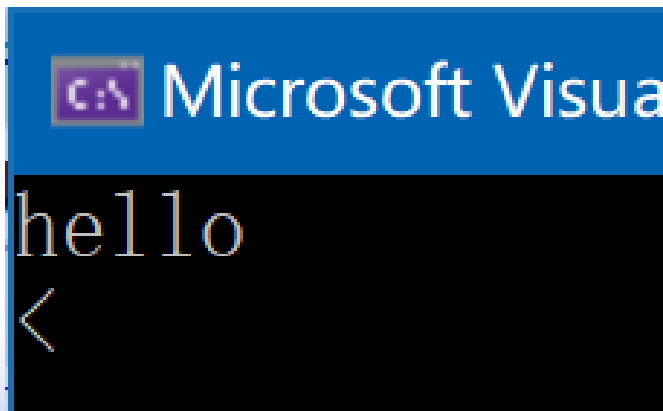
- 何时进行了越界检查？
- 如何进行越界检查？
- 能否发生了越界，而系统又未检测出来？
- 在没有越界检查时，数组越界是否不会导致程序崩溃？





小测验——函数返回局部地址

```
#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;
}
```

i
a[20]
p

temp[20]
.....
i
a[20]
p

printf传入
的参数
.....
i
a[20]
p



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



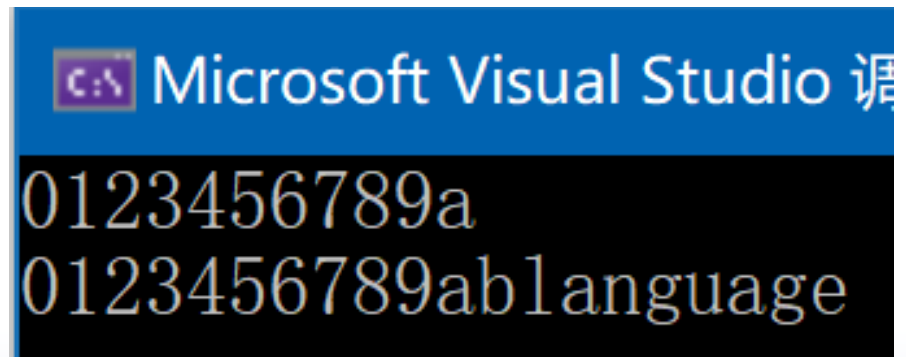


小测验——字符串访问

```
#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;
}
```

字符串 以 0 为结束符

Q: 运行结果?



Microsoft Visual Studio 调

```
0123456789a
0123456789ablanguage
```

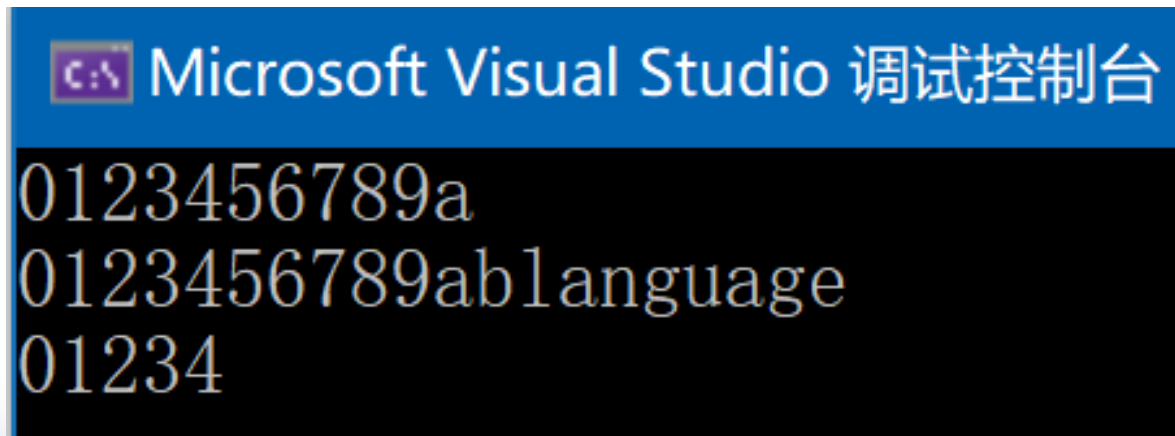


常见错误 字符串

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

字符串 以 0为结束符

Q: 运行结果?



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

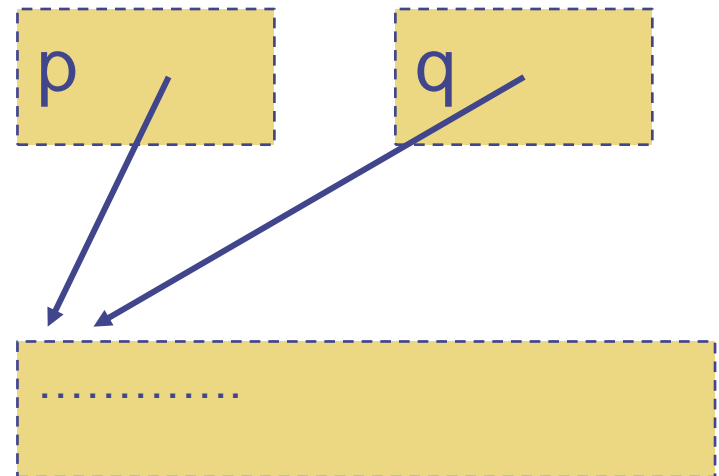




小测验——空间的分配与释

```
#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;
}
```

Q: 程序运行如何?



程序运行异常





空间的分配与释放

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

Q: 程序存在何种问题?

内存泄露





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.
```





常见错误

数组越界

返回函数中局部变量的地址

空间分配与释放不匹配

指针指向的地址错误





华中科技大学

学习要点

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

表达式的类型检查





变量的三个属性

- 数据类型
- 单元地址
- 单元内容

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

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\本科教
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    x=0x31323334;
```

x int 类型

&x int * 类型 *(&x) int 型

(char *)&x char * 类型

由int * 强制转换而来

*(char *)&x char 类型

名称	值	类型
x	825373492	int
▸ &x	0x008ffa20 {825373492}	int *
▸ (char *)&x	0x008ffa20 "4321烫烫..."	char *
*(char *)&x	52 '4'	char
*(short *)&x	13108	short
*(int *)&x	825373492	int

➤ 变量类型

➤ 变量地址的
类型

➤ 地址类型的
转换

0x34	0x8ffa20
0x33	0x8ffa21
0x32	0x8ffa22
0x31	0x8ffa23

0x3334 = 13108

((char)&x + 1) 0x33



变量的三个属性

```
int    x=0x31323334;
```

```
char* cp = (char *)&x;
```

```
short* sp = (short *)&x;
```

```
int*    ip =&x;
```

```
int     y = (int) &x;
```

➤地址类型的转换

将一个地址 转换成一个整型数

从 int * → int

名称	值	类型
❏ x	825373492	int
▶ ❏ &x	0x0053fd84 {825373492}	int *
▶ ❏ cp	0x0053fd84 "4321烫... 🔍 ▼	char *
▶ ❏ sp	0x0053fd84 {13108}	short *
▶ ❏ ip	0x0053fd84 {825373492}	int *
❏ *cp	52 '4'	char
❏ *sp	13108	short
❏ y	5504388	int
❏ y,x	0x0053fd84	int

int *p=(int *)0x0053fd84; 从 int → int *



变量的三个属性

```
int    x=0x31323334;
float  u = 1.25;
int    v = (int )u; // float → int
int    w = *(int*)&u;
float  fx = x; // int → float
float  fy = *(float*)&x;
int → int *   → float *   → float
    x      &x   (float *)&x  *(..*)&x
```

名称	值	类型
u	1.25000000	float
v	1	int
w	1067450368	int
x	825373492	int
fx	825373504.	float
fy	2.59315147e-09	float

地址类型的转换

VS 数据类型的转换

从int → float或相反，
数值大小保持不变，但两
种值的表示方法不同。

注意，转换精度有变化

将一个单元中的0-1串，当
做整数看，与当成浮点数
看，结果是不同的。

Q : $w=*(int *)\&u;$ 转换过程是什么？怎样解释转换结果？





变量的三个属性

```
int x=0x31323334;
```

Q: 能否写表达式 $\&(\&x)$?

$\&$ 要求左值

$\&x$ 表示一个地址，变成了一个数，无法再取该数的地址。

$\&(*(\&x)) == \&x$

关键

定义一个变量，
给变量分配了空间，
该空间的地址不能改变。

Q: 能否 $\&x = 0x008ffa24$; ?

不能将 `int` 转换为 `int *`

Q: 能否 $\&x = (\text{int} *)0x008ffa24$; ?

=左操作数必须为左值

表达式必须是可修改的左值





变量的三个属性

总结

- 对一个int 类型的变量x, &x 为 int *, 表示其地址;
&x 是一个右值, &x 是无法修改的;
- 一个地址是有类型的, 通过 (char *), (short *), (int *), (float *), (double *), (任意类型 *), 转换为其他类型;
- 根据一个地址去访问对应单元内容时, 是要指明地址类型的;
x 等价于 *(int *)&x;
- 相同地址, 当成不同类型来看, 虽然单元中的内容不变, 但解读出来的结果不同;

$*(char *)&x \neq *(short *)&x \neq *(float *)&x$

- 一个整数数值 可以转换为某种地址类型, 反之亦然;

$0x12345678 \rightarrow (int *)0x12345678 \quad int \rightarrow int *$

$\&x \rightarrow (int *)&x \quad int * \rightarrow int$

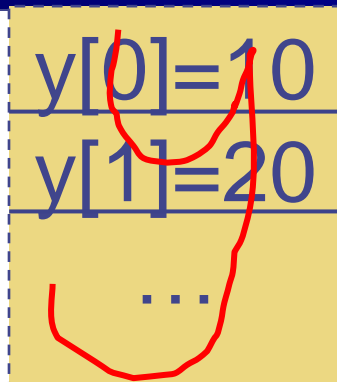




华中科技大学

变量的三个属性

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



&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;
```

$p[i] == *(p+i) == y[i]$

68 % 未找到相关问题

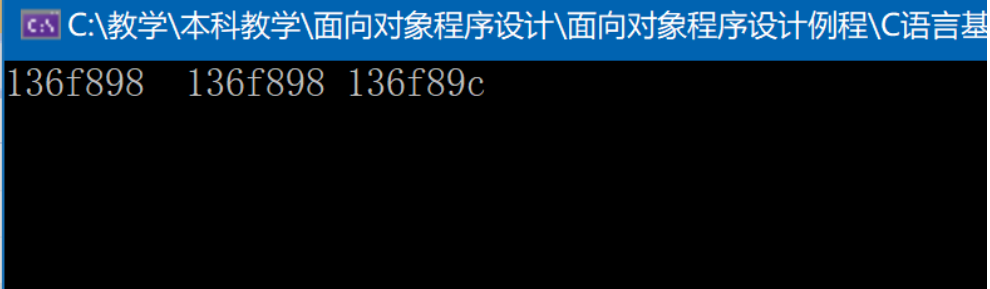
监视 1

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 *



$y[0]$ 为int类型; y 为 $\text{int}[5]$ 类型, 等同 $\text{int} *$
 $\&y[0]$ 为 $\text{int} *$ 类型; $\&y$ 为 $\text{int} (*)[5]$ 类型



变量的三个属性

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

```
int x;
```

```
int *p;
```

10	y[0]
20	y[1]
...	y[2]

0x0136f898

0x0136f89c

0x0136f8a0

单元 x, y[0], y[1], y[2],

地址 &x, &y[0], &y[1], &y[2],

Q: p=p+1;时,
p的值加了多少?
有何规律?

y[i] 的地址, 是 y的地址 + i*元素的长度

p= y; ↔ p=&y[0];

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

&p

p=0x136f898

&p

p=0x136f89c



变量的三个属性

一维数组作为 参数

int y[5];

10	y[0]
20	y[1]
	y[2]
...	

0x0136f898

0x0136f89c

0x0136f8a0

z=0x136f898

函数调用: f(y); 传递的参数只有 数组的起始地址
0x0136f898

函数定义:

```
void f(int *z) { .....} // int *z=y;  
void f(int z[10]) { .....} // int *z=y;  
void f(int z[ ]) { .....} // int *z=y
```





变量的三个属性 一维数组总结

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

- 每一个元素，如 $y[0]$ ，都是 `int` 类型， $\&y[0]$ 是 `int *` 类型；
- y 是 `int[5]` 类型，可以直接当成 `int *` 类型来看；
- $\&y$ 是 `int (*)[5]` 类型；
- 用代表的地址值来看， $\&y[0] = y = \&y$ ；
- 地址值可以转换为相同类型，

$\&y[0] = (\text{int} *)\&y[0] = (\text{int} *)y = (\text{int} *)\&y$

- 指针可以当成数组使用；数组可以当成指针用

`int *p; p[i] = *(p+i) y[i] = *(y+i)`



变量的三个属性

二维数组，数据存放的规律，各种写法对应的数据类型

```
int z[3][4] = { {10, 20, 30, 40}, {5, 6, 7, 8} };
```

监视 1		
搜索(Ctrl+E) 搜索深度: 3		
名称	值	类型
z[0][0]	10	int
z[0]	0x00cffa18 {10, 20, 30, 40}	int[4]
z[1]	0x00cffa28 {5, 6, 7, 8}	int[4]
z[2]	0x00cffa38 {0, 0, 0, 0}	int[4]
z	0x00cffa18 {0x00cffa18 {10, 20, 30, 40}, ...}	int[3][4]
[0]	0x00cffa18 {10, 20, 30, 40}	int[4]
[1]	0x00cffa28 {5, 6, 7, 8}	int[4]
[2]	0x00cffa38 {0, 0, 0, 0}	int[4]

z[0][0] → int

z[0] → int [4] or int *

z → int [3][4] or int (*)[4]

特别注意;

z 与 int ** 的差异

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], z[1][0],

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

地址 z[0], z[0]+1, z[0]+2, z[1],

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[0][1];

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

p = (int *)z; 三者的功能等价

变量的三个属性

Q: 如何定义 `int[4] *` 类型的指针?

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

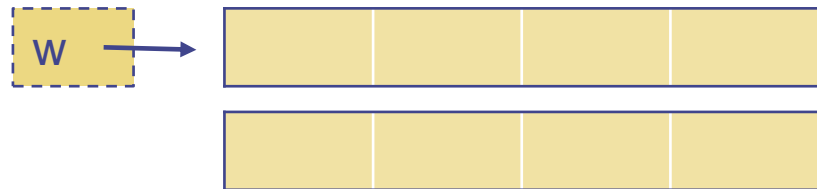
```
int *p;
```

```
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 = &(z[0]);      z[0] -> int[4]
```

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

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

类型的正确写法
`int (*) [4]`

```
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

```
int (* p2)[4];  
int z[3][4] = { {10, 20, 30, 40}, {5, 6, 7, 8} };  
p2 = &(z[0]);
```

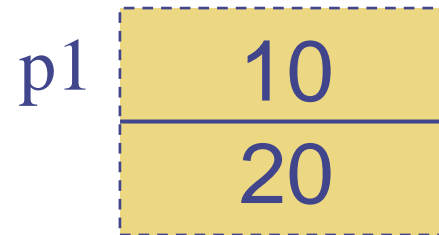
监视 1		
搜索(Ctrl+E) 🔍 ⏪ ⏩ 搜索深度: 3 ▼ ▾ A		
名称	值	类型
▶ 📦 z[0]	0x010ff850 {10, 20, 30, 40}	int[4]
▶ 📦 &z[0]	0x010ff850 {10, 20, 30, 40}	int[4] *
▶ 📦 p2	0x010ff850 {10, 20, 30, 40}	int[4] *

```
int *p3;  
p3 = z[0];
```






变量的三个属性

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



```
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



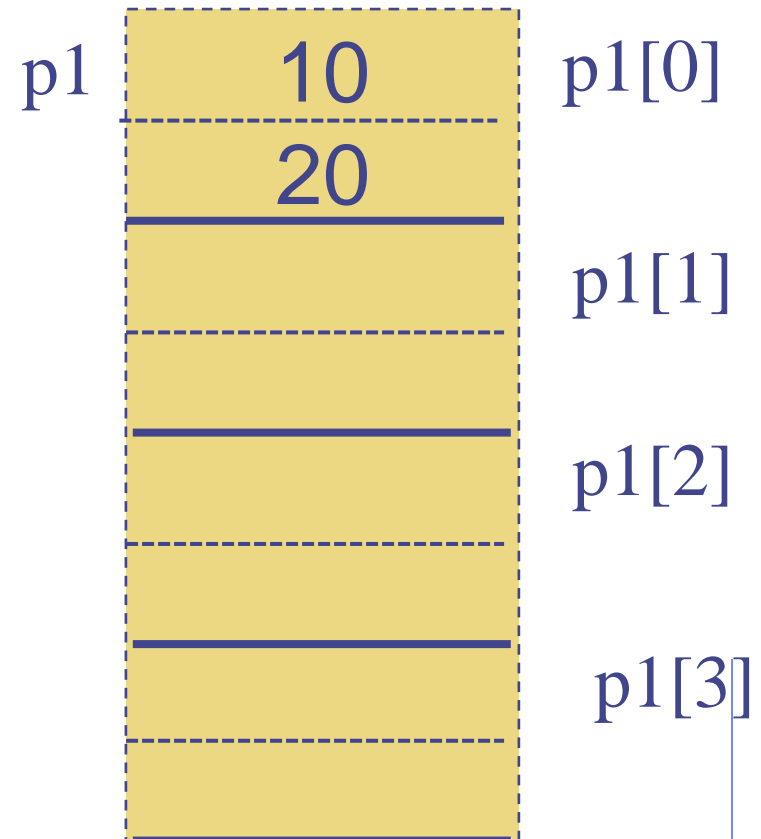


变量的三个属性

```
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

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;
```

0x005efb20

x=10

```
*p=20;
```

```
p[0]=20;
```

```
*(p+0)=20;
```

```
int *p;
```

```
p=&x;
```

0x005efb34

p=

0x005ebf20

p

0x005efb34

指针和一维数组用法的等价性





指针变量

```
int x;
```

```
x=10;
```

```
int *p1;
```

```
p1=&x;
```

0x005efb20

x=10

0x005ebf20

p1

0x005efb34

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

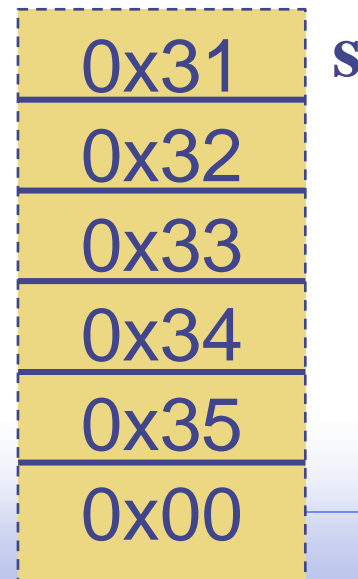
```
char s[10] = "12345";  
printf("%c %x\n", *s, *s);  
printf("%d %x\n", *(short *)s, *(short *)s);  
printf("%d %x\n", *(int *)s, *(int *)s);
```

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);
```





总结

一个变量（不论简单类型、复合类型）的三个要素

- 变量的数据类型
- 变量的地址
- 变量（单元）中的内容

强制类型转换 由一种类型变成另一种类型

（类型表达式）数值表达式

类型示例 `int`、`int *`、`int **`、`int (*) [n]`

地址类型转换 VS 数据类型转换





作业 1

定义了 结构 student ， 以及结构数组变量s[3];

```
struct student {  
    char name[8];  
    short age;  
    float score;  
    char remark[200];  
};  
student s[3];  
student new_s[3];
```

编写程序，将 s[3] 中的信息紧凑存放到 一个字符数组 message 中，然后从 message 转换到结构数组 new_s[3] 中。





作业 1

Microsoft Visual Studio 调试控制台

结构 student 的大小 =214

结构数组 s 的大小 =642

张三 20 91.5 2021年获得三好学生称号, 2020年获得优秀干部奖

xuxiang 21 94.7 very good student

wang 22 87.6 none

packed :103

张三 20 91.5 2021年获得三好学生称号, 2020年获得优秀干部奖

xuxiang 21 94.7 very good student

wang 22 87.6 none

要求 s[0].name 为自己的姓名;

可以在给的程序中 补充

void pack_student(student* s)

void restore_student(student* s)

