

§ 6. 指针与引用

6.2. 字符串与指针

6.2.1. 字符指针的定义及使用 (重复之前的内容, 基类型为char/unsigned char)

char *指针变量名

char *p;

指向简单变量的指针:

char ch='A', *p;

p=&ch; 地址

赋值语句

char ch, *p=&ch; 定义时赋初值

*p='B'; 值, 相当于ch='B'

指向一维数组的指针:

char ch[10], *p;

p=&ch[3]; 指向数组的第3个元素

p=ch / p=&ch[0]; 指向数组的开始

指向二维数组的指针:

char ch[10][20];

char *p1; //指向元素的指针

char (*p2)[20]; //指向一维数组的指针

p1 = &ch[0][0]; / p1 = ch[0]; / p1 = *ch;

p2 = ch; / p2 = &ch[0];

§ 6. 指针与引用

6.2. 字符串与指针

6.2.2. 用字符指针指向字符串

数组模块中:

```
char s[]="china"; //一维字符数组, 缺省为6
```

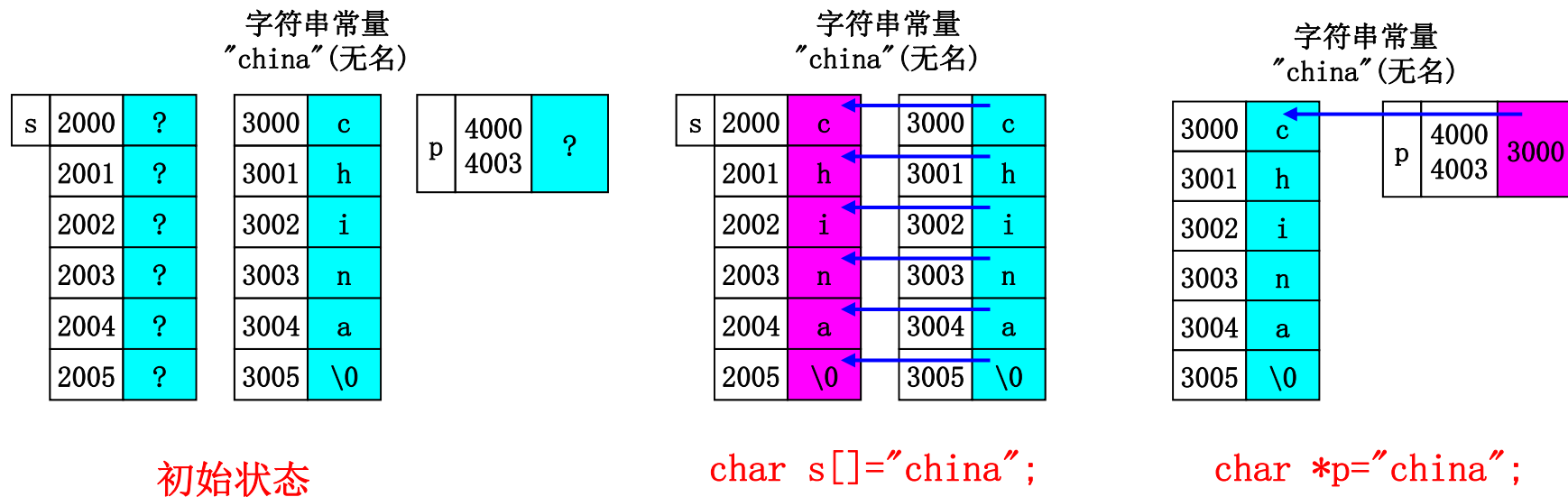
```
string s="china"; //C++类变量表示, 暂不讨论
```

本模块:

```
char *p="china"; //字符指针指向字符串
```

区别: s是一个一维字符数组名, 占用一定的内存空间, 编译时确定地址不变 (s++错)

p是基类型为char的指针变量, 表示一个字符(串首字符)的地址, 在程序的运行中可以改变



§ 6. 指针与引用

6.2. 字符串与指针

6.2.2. 用字符指针指向字符串

定义时赋初值

```
char *p="china"; ✓
```

用赋值语句赋值

```
char *p;
```

```
p="china"; ✓
```

p表示取地址，将字符串常量的首地址赋给p

```
*p="china"; ✗
```

*p表示取值，基类型是char，因此不能是字符串

```
*p='c'; ?
```

编译正确，能否正确执行视情况而定，具体例子后面会给出

是否正确?
后续解决

定义时赋初值

```
char s[]="china"; ✓
```

用赋值语句赋值

```
char s[10];
```

```
s="china"; ✗
```

数组不能整体赋值

要用strcpy(s, "China");

```
*s="china"; ✗
```

*s↔s[0], char型值

```
*s='c'; ✓
```

§ 6. 指针与引用

6.2. 变量与指针

6.2.3. 字符指针的打印

```
#include <iostream>
using namespace std;
int main()
{
    short s, *p2 = &s;
    double d, *p5 = &d;

    cout << p2 << endl; 假设地址A
    cout << ++p2 << endl; =地址A+2 2字节
    cout << p5 << endl; 假设地址B
    cout << ++p5 << endl; =地址B+8 8字节

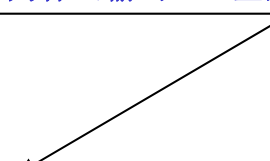
    return 0;
}
```

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    short s, *p2 = &s;
    char d, *p5 = &d;
    cout << p2 << endl;
    cout << ++p2 << endl;
    cout << hex << (int *) (p5) << endl;
    cout << hex << (int *) (++p5) << endl;

    return 0;
}
```

问题：直接用
cout << p5 << endl;
cout << ++p5 << endl;
为什么会输出一串乱字符？
为什么输出char型的地址要转为非char？

假设地址A
=地址A+2 2字节
假设地址B
=地址B+1



§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

```
char a[]="china", *p;
```

```
p=a; / p=&a[0];
```

a[i]

p[i]

*(a+i)

*(p+i)

*a++

*p++

(*a)++

(*p)++

8种表示方式中有一种
是错误的？哪种？

§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    char *p1, *p2;
    p1=str1;
    p2=str2;
    for(; *p1!='\0'; p1++, p2++)
        *p2=*p1;
    *p2='\0';
    p1=str1;
    p2=str2;
    cout << "str1:" << p1 << endl
           << "str2:" << p2 << endl;
    return 0;
}
```

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    int i;
    for(i=0; str1[i]!='\0'; i++)
        str2[i] = str1[i];
    str2[i]='\0';
    cout << str1 << endl;
    cout << str2 << endl;
    return 0;
}
```

数组法实现

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    int i;
    for(i=0; *(str1+i]!='\0'; i++)
        *(str2+i) = *(str1+i);
    *(str2+i]='\0';
    cout << str1 << endl;
    cout << str2 << endl;
    return 0;
}
```

等价指针法

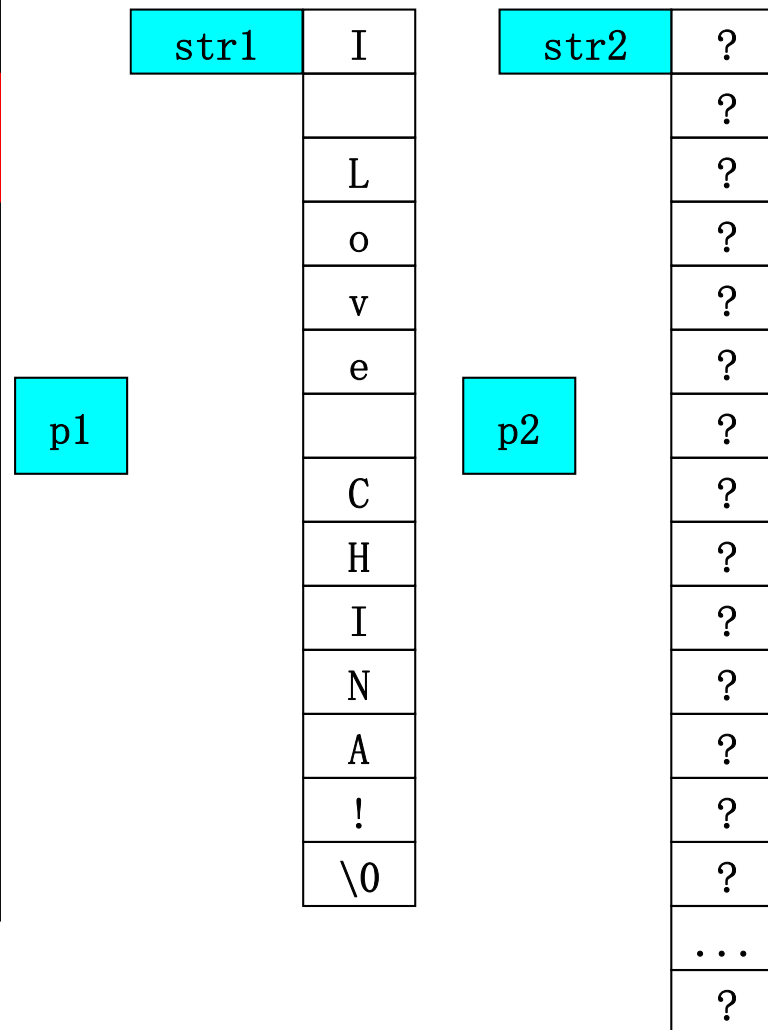
§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    char *p1, *p2;
    p1=str1;
    p2=str2;
    for(; *p1!='\0'; p1++, p2++)
        *p2=*p1;
    *p2='\0';
    p1=str1;
    p2=str2;
    cout ...
    return 0;
}
```



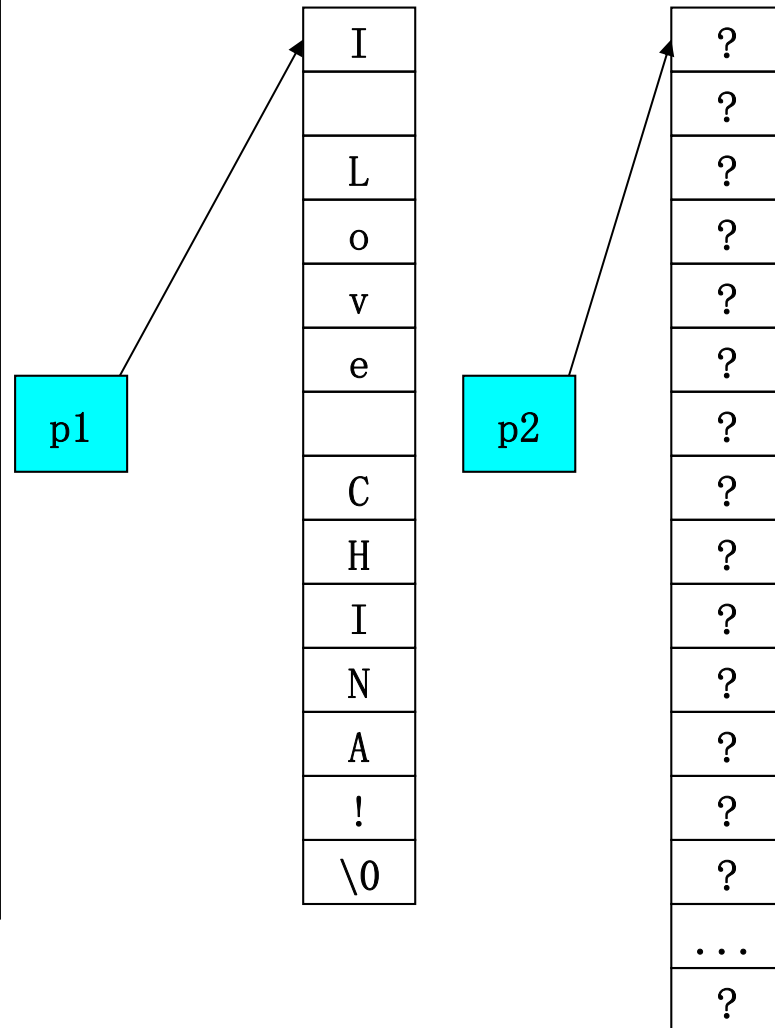
§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{ char str1[]="I Love CHINA!", str2[20];
  char *p1,*p2;
  p1=str1;
  p2=str2;
  for(;*p1!='\0';p1++,p2++)
    *p2=*p1;
  *p2='\0';
  p1=str1;
  p2=str2;
  cout ...
  return 0;
}
```



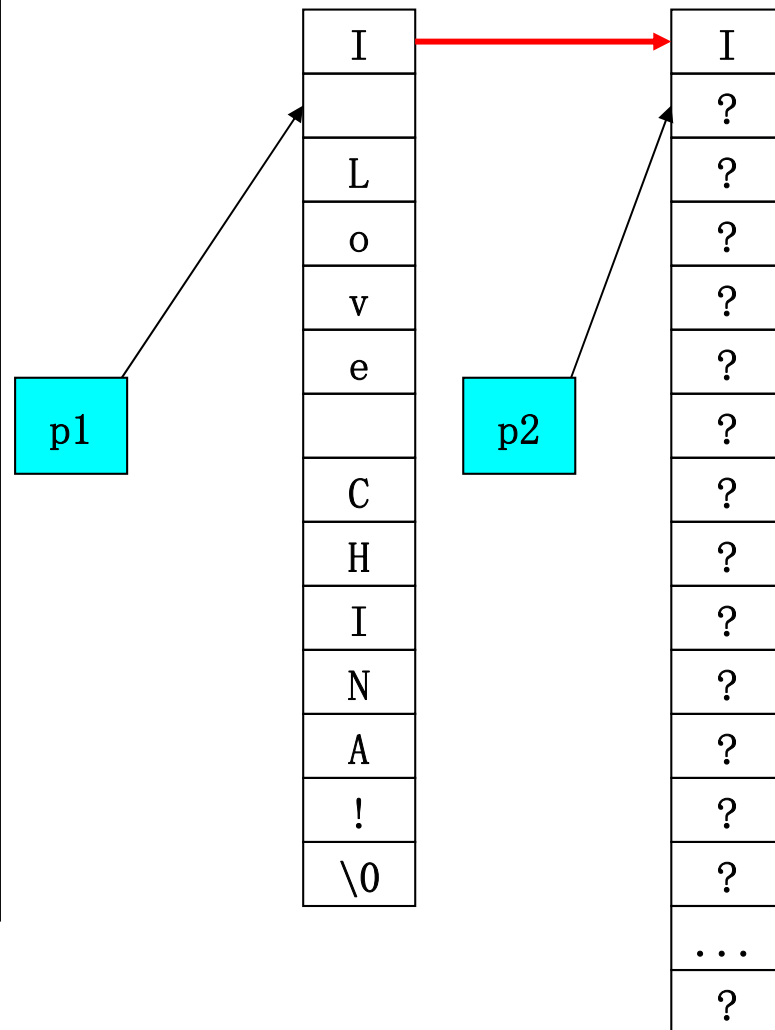
§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{ char str1[]="I Love CHINA!", str2[20];
  char *p1,*p2;
  p1=str1;
  p2=str2;
  for(;*p1!='\0';p1++,p2++)
    *p2=*p1;
  *p2='\0';
  p1=str1;
  p2=str2;
  cout ...
  return 0;
}
```



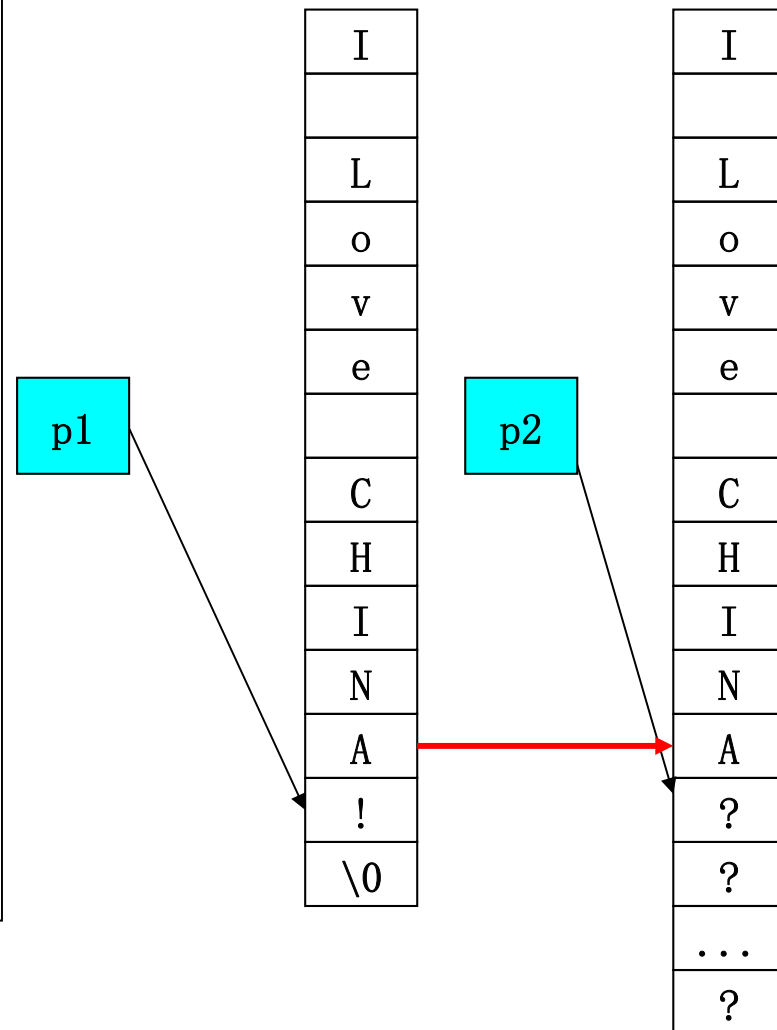
§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    char *p1,*p2;
    p1=str1;
    p2=str2;
    for(;*p1!='\0';p1++,p2++)
        *p2=*p1;
    *p2='\0';
    p1=str1;
    p2=str2;
    cout ...
    return 0;
}
```



§ 6. 指针与引用

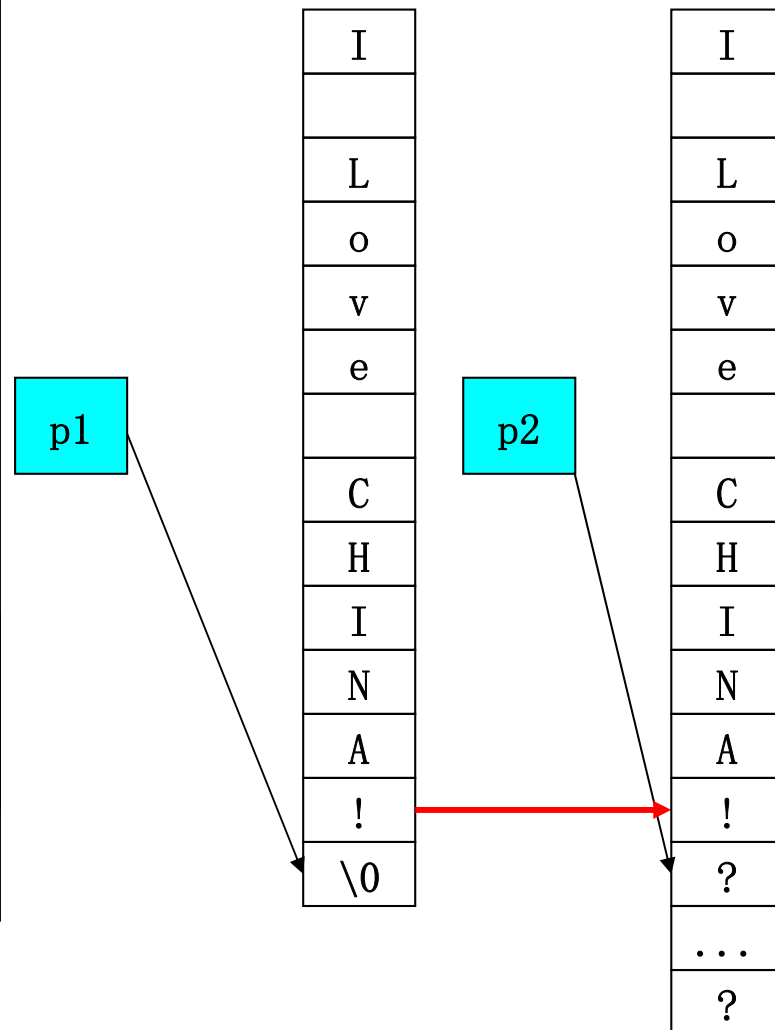
6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{ char str1[]="I Love CHINA!", str2[20];
  char *p1,*p2;
  p1=str1;
  p2=str2;
  for(;*p1!='\0';p1++,p2++)
    *p2=*p1;
  *p2='\0';
  p1=str1;
  p2=str2;
  cout ...
  return 0;
}
```

！赋值完成后，
循环结束
注意：\0未赋



§ 6. 指针与引用

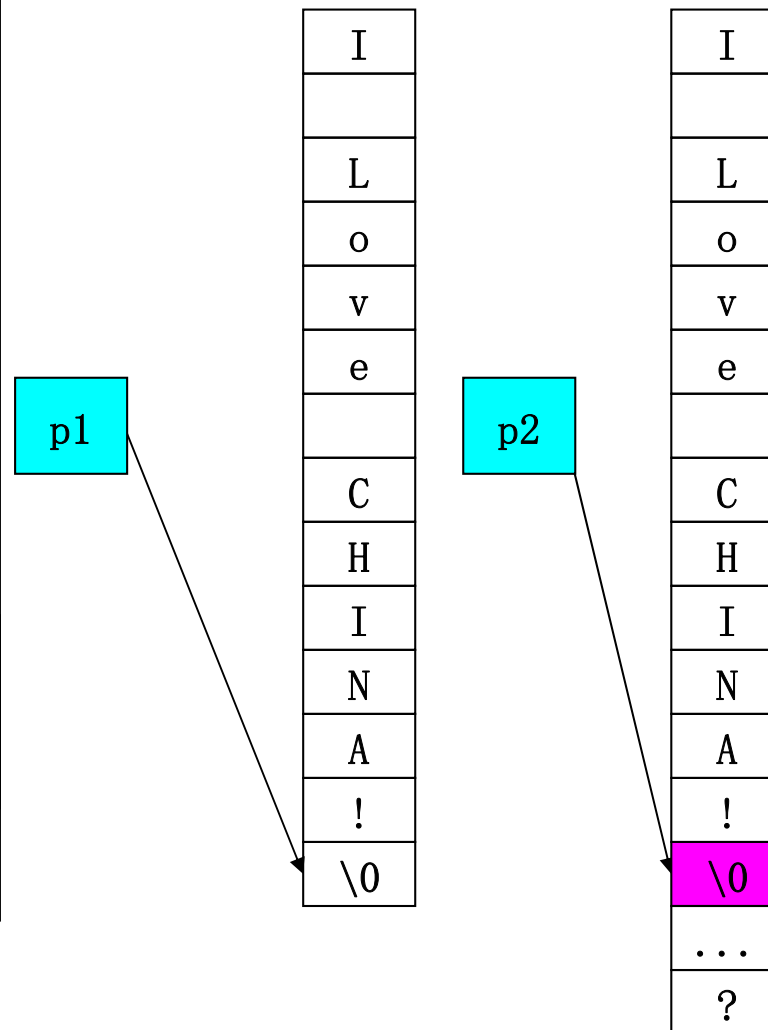
6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    char *p1,*p2;
    p1=str1;
    p2=str2;
    for(;*p1!='\0';p1++,p2++)
        *p2=*p1;
    *p2='\0';
    p1=str1;
    p2=str2;
    cout ...
    return 0;
}
```

！赋值完成后，
循环结束
注意：\0未赋



§ 6. 指针与引用

6.2. 字符串与指针

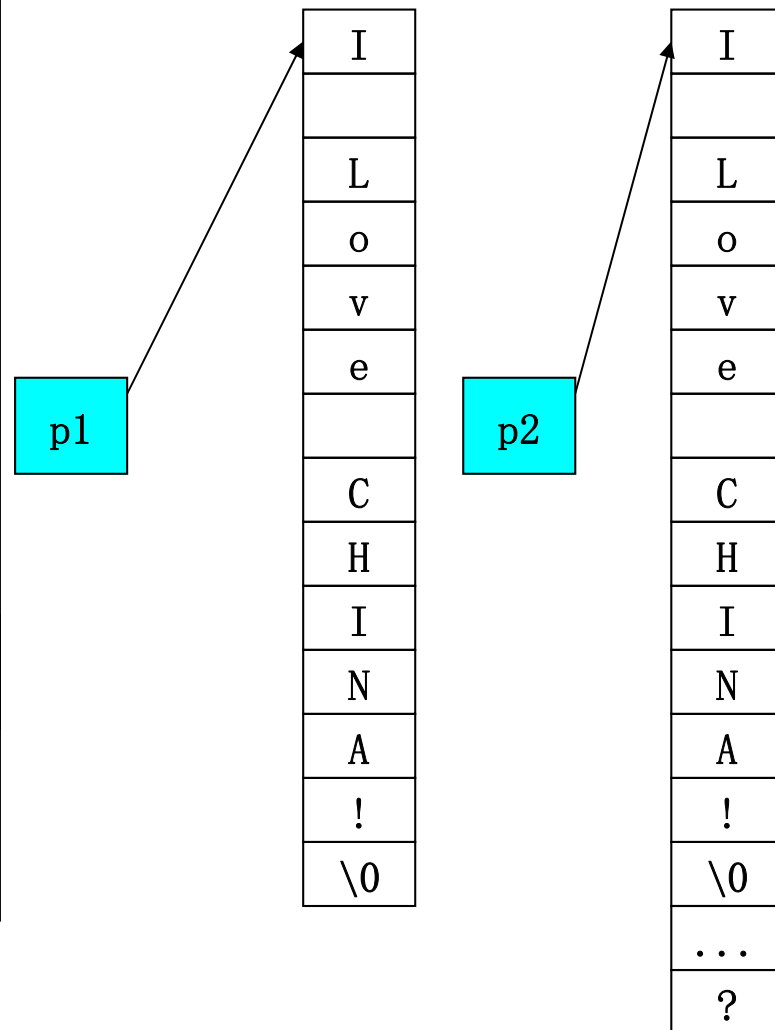
6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    char *p1,*p2;
    p1=str1;
    p2=str2;
    for(;*p1!='\0';p1++,p2++)
        *p2=*p1;
    *p2='\0';
    p1=str1;
    p2=str2;
    cout ...
    return 0;
}
```

重新指向
并打印

Microsoft Visual Studio 调试控制台
str1=I Love CHINA!
str2=I Love CHINA!



§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{ char str1[]="I Love CHINA!", str2[20];
  char *p1,*p2;
  p1=str1;
  p2=str2;
  for(;*p1!='\0';p1++,p2++)
    *p2=*p1;
  *p2='\0';
  p1=str1;
  p2=str2;
  cout ...
  return 0;
}
```

```
for(;*p1!='\0';)
    *p2++ = *p1++;
```

```
for(;*p1;)
    *p2++ = *p1++;
```

```
while(*p1)
    *p2++ = *p1++;
```

1、判断
2、赋值
3、++到下一元素
循环结束,\0未赋

几种等价表示

§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{ char str1[]="I Love CHINA!", str2[20];
  char *p1,*p2;
  p1=str1;
  p2=str2;
  while ((*p2=*p1) != '\0') {
    p1++; p2++;
  }
  *p2='\0';
  p1=str1;
  p2=str2;
  cout ...
  return 0;
}
```

1、赋值
2、判断
3、++到下一元素
循环结束, \0已赋

虽不完全等价
但都是正确的

```
for (; *p1 != '\0'; p1++, p2++)
    *p2 = *p1;
```

1、判断
2、赋值
3、++到下一元素
循环结束, \0未赋

```
while (*p2 = *p1) {
    p1++; p2++;
}
```

1、赋值
2、++到下一元素
3、判断旧值
循环结束, \0已赋

```
while (*p2++ = *p1++);
```

```
while (*p2++ = *p1++)
    ;
```

§ 6. 指针与引用

6.2. 字符串与指针

6.2.4. 下标法与指针法处理字符串

例：将str1的内容复制给str2

```
int main()
{
    char str1[]="I Love CHINA!", str2[20];
    char *p1, *p2;
    p1=str1;
    p2=str2;
    do {
        *p2++ = *p1++;
    } while(*p1);
    *p2='\0';
    p1=str1;
    p2=str2;
    cout ...
    return 0;
}
```

- 1、赋值
 - 2、++到下一元素
 - 3、判断新值
- 循环结束, \0未赋

问：改为do-while循环后是否正确？

答：就本例而言，是正确的

若 `char str1[]=""`，即空串的情况下
有可能错误

- 1、`str2[0] = str1[0]`
`'\0' <= '\0'`
- 2、++到 `str1[1]`
- 3、判断`str1[1]`是否`\0`

至此，`str1[1]`已越界，但程序会继续执行

若`str1[1]!='\0'`，则

`str2[1] = str1[1]`

依次类推到`str1[x]`是`'\0'`为止

若`x<=19`，则读非法，但写仍在`str2`的合理
范围内，错误可能不体现

若`x>19`，则读非法，且对`str2`的赋值会
超过数组长度20，会导致非法写，
出错概率大于`x<=19`

§ 6. 指针与引用

6.2. 字符串与指针

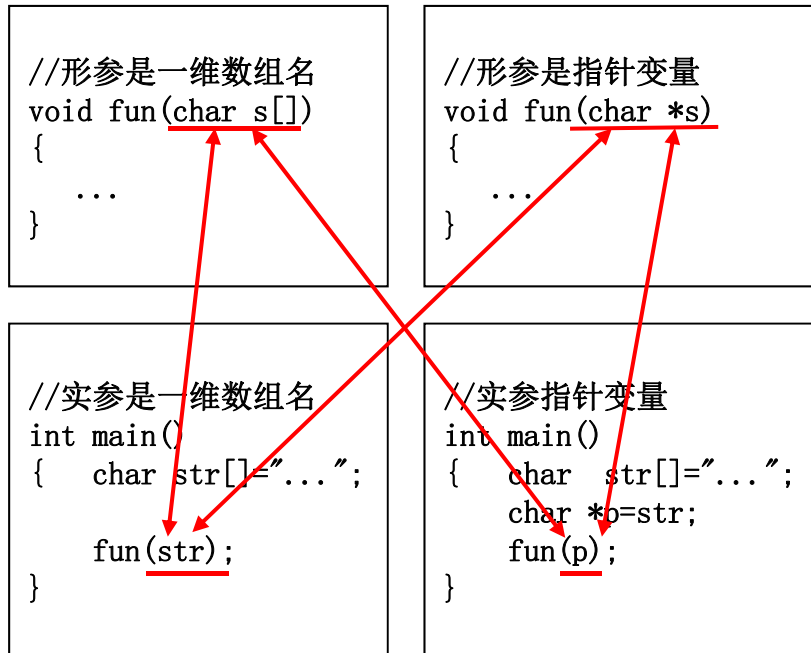
6.2.4. 下标法与指针法处理字符串

<pre>#include <iostream> using namespace std; int main() { char str1[]="I love CHINA!", str2[20]; char *p1 = str1, *p2 = str2; cout << hex << (int *) (str2) << endl; for(;;*p1!='\0';p1++,p2++) *p2=*p1; *p2='\0'; cout << hex << (int *) (p2) << endl; cout << dec << p2-str2 << endl; cout << str2 << endl; }</pre>	<p>完全正确 地址str2</p> <p>地址str2+13 13 I love CHINA!</p>	<pre>#include <iostream> using namespace std; int main() { char str1[]="", str2[20]; char *p1 = str1, *p2 = str2; cout << hex << (int *) (str2) << endl; for(;;*p1!='\0';p1++,p2++) *p2=*p1; *p2='\0'; cout << hex << (int *) (p2) << endl; cout << dec << p2-str2 << endl; cout << str2 << endl; }</pre>	<p>完全正确 地址str2</p> <p>地址str2 0 空行</p>
<pre>#include <iostream> using namespace std; int main() { char str1[]="I Love CHINA!", str2[20]; char *p1 = str1, *p2 = str2; cout << hex << (int *) (str2) << endl; do { *p2++ = *p1++; } while(*p1); *p2='\0'; cout << hex << (int *) (p2) << endl; cout << dec << p2-str2 << endl; cout << str2 << endl; }</pre>	<p>目前正确 地址str2</p> <p>地址str2+13 13 I love CHINA!</p>	<pre>#include <iostream> using namespace std; int main() { char str1[]="", str2[20]; char *p1 = str1, *p2 = str2; cout << hex << (int *) (str2) << endl; do { *p2++ = *p1++; } while(*p1); *p2='\0'; cout << hex << (int *) (p2) << endl; cout << dec << p2-str2 << endl; cout << str2 << endl; }</pre>	<p>运气好正确 本质不正确!!! 地址str2</p> <p>地址str2+x x (不确定) 空行</p>

§ 6. 指针与引用

6.2. 字符串与指针

6.2.5. 指向字符数组的指针作函数参数 (四种组合, 同前)



§ 6. 指针与引用

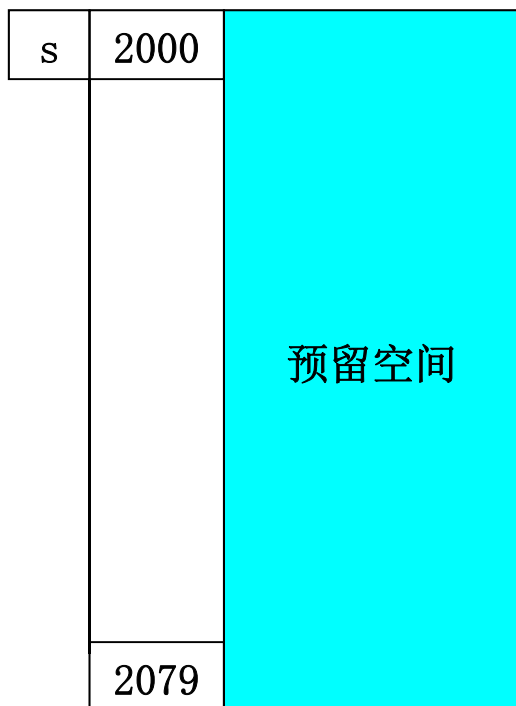
6.2. 字符串与指针

6.2.6. 字符指针与字符数组的区别

★ 字符数组占用一定的连续存储空间，而指针仅有一个有效地址的存储空间

```
char s[80];  
cin >> s;
```

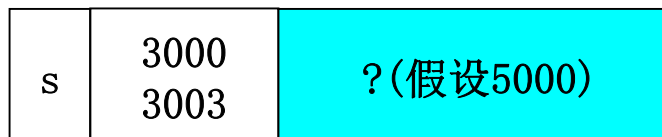
当输入小于80个字符时，正确



```
char *s;  
cin >> s;
```

VS2019编译报error

其他编译器能运行，但运行出错



设键盘输入10个字符，则会覆盖
5000-5010的11字节的空间(非法占用)

如何使正确?

```
char ss[80], *s;  
s=ss; //使s指向确定空间  
cin >> s;
```

§ 6. 指针与引用

6.2. 字符串与指针

6.2.6. 字符指针与字符数组的区别

★ 字符数组占用一定的连续存储空间，而指针仅有一个有效地址的存储空间

★ 赋初值的方式相同，但含义不同

```
char s[]="china";
```

```
char *p="china";
```

字符串常量 "china"(无名)		
s	2000	c
	2001	h
	2002	i
	2003	n
	2004	a
	2005	\0

p	4000 4003	3000
---	--------------	------

★ 字符数组在执行语句中赋值时只能逐个进行，而字符指针仅能整体赋首址

```
char s[80];
```

```
s="china"; ✗
```

数组不允许整体赋值

```
s[0]='c'; ✓
```

```
s[1]='h'; ✓
```

```
s[2]='i'; ✓
```

```
s[3]='n'; ✓
```

```
s[4]='a'; ✓
```

```
s[5]='\0'; ✓
```

s	2000	预留空间
	2079	

```
char *s;
```

```
s="china"; ✓
```

```
s[0]='c'; ✗ 非法占用
```

```
s[1]='h'; ✗
```

```
s[2]='i'; ✗
```

```
s[3]='n'; ✗
```

```
s[4]='a'; ✗
```

```
s[5]='\0'; ✗
```

s	3000 3003	?(假设5000)
---	--------------	-----------

```
char ss[80], *s;  
s=ss;  
指向确定地址后正确
```

★ 数组首地址的值不可变，指针的值可变

§ 6. 指针与引用

6.2. 字符串与指针

6.2.6. 字符指针与字符数组的区别

6.2.2 中的遗留问题:

定义时赋初值

```
char *p="china"; ✓
```

用赋值语句赋值

```
char *p;
```

```
p="china"; ✓ p表示取地址, 将字符串常量的首地址赋给p
```

```
*p="china"; ✗ *p表示取值, 基类型是char, 因此不能是字符串
```

```
*p='c'; ? 编译正确, 能否正确执行视情况而定, 具体例子后面会给出
```

是否正确?
后续解决

```
char *p; *p='c';
```

正确/错误的各种情况

```
int main()
{
    char *p;
    *p='c';
    return 0;
} //编译有警告, 运行错
p指向未确定空间
```

```
int main()
{
    char ch, *p=&ch;
    *p='c';
    return 0;
} //正确, ch被赋值为'c'
```

```
int main()
{
    char *p=(char *)"Hello";
    *p='c';
    return 0;
} //编译不错运行错
p已指向常量
```

```
int main()
{
    char ch[]="Hello";
    char *p=ch;
    *p='c';
    return 0;
} //正确, ch变为"cello"
```

§ 6. 指针与引用

6.3. 引用 (C++新增)

6.3.1. 引用的基本概念

含义：变量的别名

声明：int a, &b=a; //a和b表示同一个变量

★ 引用不分配单独的空间 (指针变量有单独的空间)

变量的定义：分配空间

变量的声明：不分配空间

★ 引用需在声明时进行初始化，指向同类型的变量，在整个生存期内不能再指向其它变量

```
int a, &c=a; //正确
int b, &c=b; //错
      &c=b; //错
c已是a的别名, 不能再
无论定义/赋值均不行
```

```
int a, &c=a, b;
      c=b/b=c ⇔ a=b/b=a
int a, &c=a, b[10];
      c=b[3] ⇔ a=b[3]
都正确
```

★ 不能声明引用数组和指向引用的指针，但可声明数组的引用、数组元素的引用和指向指针的引用

```
int &b[3];           //错误，不能声明引用数组
int &*p;             //错误，不能定义指向引用的指针
int a[5], (&b)[5]=a; //正确，引用指向整个数组
int a[5], &b=a[3];   //正确，引用指向数组元素
int *a, *&b=a;      //正确，指向指针的引用
```

§ 6. 指针与引用

6.3. 引用 (C++新增)

6.3.1. 引用的基本概念

含义：变量的别名

声明：int a, &b=a; //a和b表示同一个变量

★ 引用不分配单独的空间 (指针变量有单独的空间)

★ 引用需在声明时进行初始化，指向同类型的简单变量，在整个生存期内不能再指向其它变量

★ 不能声明指向数组的引用、引用数组和指向引用的指针，但可声明数组元素的引用和指向指针的引用

★ &的理解

定义语句新变量名前：引用声明符

```
int a, &b=a;
```

其它 (定义语句已定义变量名，执行语句)： 取地址运算符

```
int a, *p=&a;
```

```
p=&a;
```

引用的简单使用：出现在普通变量可出现的任何位置

右侧例子无任何实用价值

1、多定义一个名称

2、两者容易混淆

//例：简单变量的引用

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{   int a=10, &b=a;
```

```
    a=a*a;
```

```
    cout << a << " " << b << endl;    100  100
```

```
    b=b/5;
```

```
    cout << a << " " << b << endl;    20   20
```

```
    return 0;
```

```
}
```

§ 6. 指针与引用

6.3. 引用 (C++新增)

6.3.1. 引用的基本概念

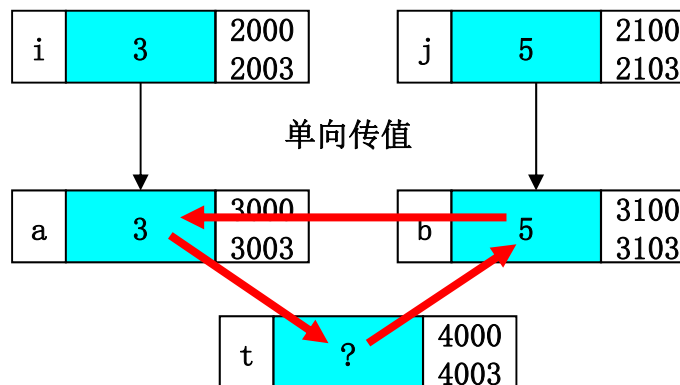
6.3.2. 引用作函数参数

例：两数交换

```
void swap(int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
}

int main()
{
    int i=3, j=5;
    swap(i, j);
    cout << i << " " << j << endl;
    return 0;
}
```

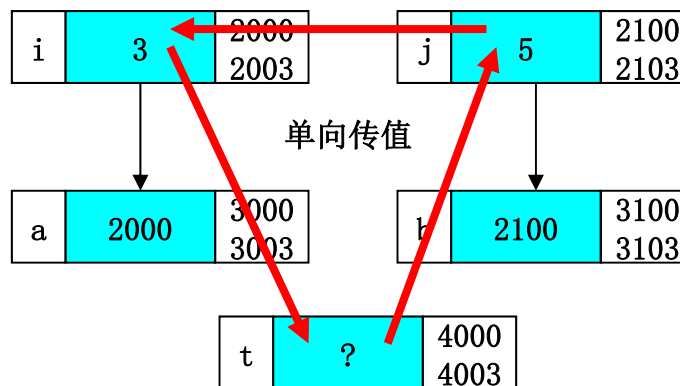
直接传值
错误



```
void swap(int *a, int *b)
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}

int main()
{
    int i=3, j=5;
    swap(&i, &j);
    cout << i << " " << j << endl;
    return 0;
}
```

传地址
正确



§ 6. 指针与引用

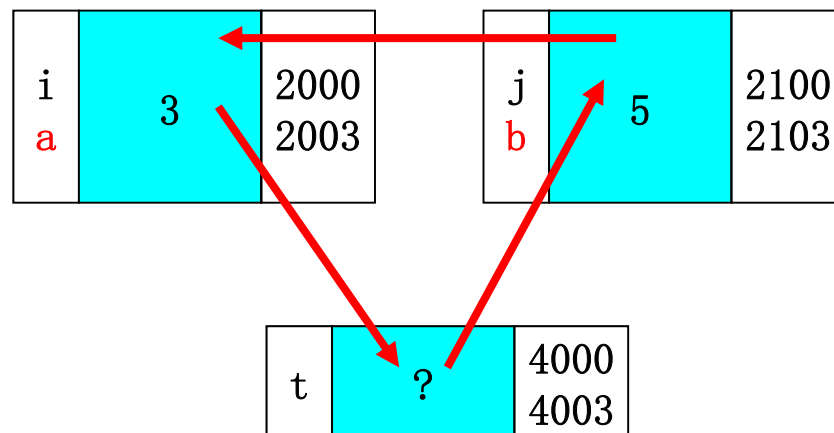
6.3. 引用 (C++新增)

6.3.1. 引用的基本概念

6.3.2. 引用作函数参数

例：两数交换

```
void swap(int &a, int &b)  引用做形参  
                           正确  
{   int t;  
    t = a;  
    a = b;  
    b = t;  
}  
  
int main()  
{   int i=3, j=5;  
    swap(i, j);  
    cout << i << " " << j << endl;  
    return 0;  
}
```



- ★ 当形参是引用时，不需要声明时初始化，调用时，形参不分配空间，只是当作实参的别名，因此对形参的访问就是对实参的访问
- ★ 实参虽然是变量名，但传递给形参的实际上是实参的地址，同时形参不单独分配空间，只是虚实结合 (地址传递方式)

§ 6. 指针与引用

6.3. 引用 (C++新增)

6.3.1. 引用的基本概念

6.3.2. 引用作函数参数

★ 实参虽然是变量名，但传递给形参的实际上是实参的地址，同时形参不单独分配空间，只是虚实结合 (地址传递方式)

● C++函数参数传递的两种方式

◆ 传值：单向传值，实形参分占不同空间

```
void fun(int x)
{ ...
}
```

形参的变化
不影响实参

```
int main()
{ int k = 10;
  fun(k);
}
```

```
void fun(int *x)
{ ...
}
```

可通过形参间接
访问实参，但本质
仍是单向传值

```
int main()
{ int k=10;
  fun(&k);
}
```

◆ 传址：实形参重合，对形参的访问就是对实参的访问

```
void fun(int *x)
{ ...
}
```

对形参数组
的修改影响
实参数组

```
int main()
{ int k[10] = {...};
  fun(k);
}
```

```
void fun(int &x)
{ ...
}
```

对形参的访问
就是对实参的访问

```
int main()
{ int k=10;
  fun(k);
}
```

§ 6. 指针与引用

6.3. 引用 (C++新增)

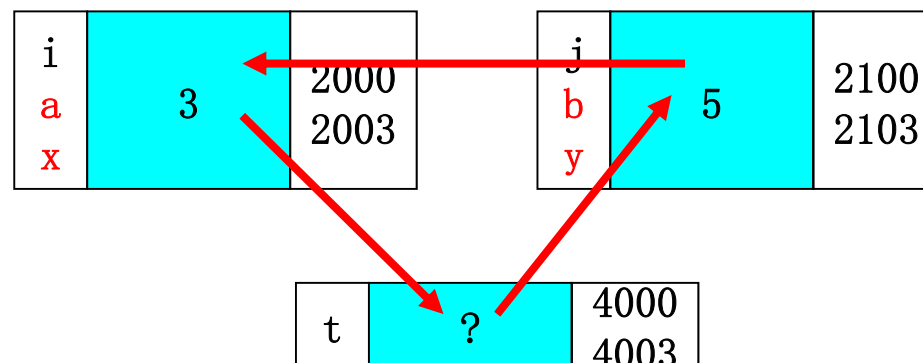
6.3.2. 引用作函数参数

- ★ 当形参是引用时，不需要声明时初始化，调用时，形参不分配空间，只是当作实参的别名，因此对形参的访问就是对实参的访问
- ★ 实参虽然是变量名，但传递给形参的实际上是实参的地址，同时形参不单独分配空间，只是虚实结合 (地址传递方式)
- ★ 引用允许传递

```
void swap1(int &x, int &y)
{
    int t;
    t = x;
    x = y;
    y = t;
}

void swap(int &a, int &b)
{
    swap1(a, b);
}

int main()
{
    int i=3, j=5;
    swap(i, j);
}
```



§ 6. 指针与引用

6.3. 引用 (C++新增)

6.3.2. 引用作函数参数

- ★ 当形参是引用时，不需要声明时初始化，调用时，形参不分配空间，只是当作实参的别名，因此对形参的访问就是对实参的访问
- ★ 实参虽然是变量名，但传递给形参的实际上是实参的地址，同时形参不单独分配空间，只是虚实结合 (地址传递方式)
- ★ 引用允许传递
- ★ 当引用做函数形参时，实参不允许是常量/表达式，否则编译错误
(形参为const引用时实参可为常量/表达式)

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

void fun(int x)
{
    cout << x << endl;
}

int main()
{
    int i=10;
    fun(i);    //正确
    fun(15);   //正确
    return 0;
}
```

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

void fun(int &x)
{
    cout << x << endl;
}

int main()
{
    int i=10;
    fun(i);    //正确
    fun(15);   //编译错
    return 0;
}
```

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

void fun(const int &x)
{
    cout << x << endl;
}

int main()
{
    int i=10;
    fun(i);    //正确
    fun(15);   //正确
    return 0;
}
```

§ 6. 指针与引用

6.3. 引用 (C++新增)

6.3.3. 关于引用的特别说明

- ★ 引用在需要**改变实参值**的函数调用时比指针方式更容易理解，形式也更简洁，不容易出错
- ★ 引用不能完全替代指针(可以将指针理解为if-else，引用理解为switch-case)
- ★ 引用是C++新增的，纯C的编译器不支持，后续工作学习中接触的大量**底层代码**仍是由C编写的，此时无法使用引用(VS/Dev都是C++编译器，兼容编译纯C，以后缀名.c/.cpp来区分如何编译)
- ★ 对于本专业而言，仍需要透彻理解指针!!!