第11讲　指　　针

薛丽萍

下面哪个程序可实现两面个数的交换

```
#include <stdio.h>
void swap(int a, int b )
{   int temp;
    temp=a; a=b; b=temp;
}

main()
{   int x=7,y=11;
    printf("x=%d,\ty=%d\n",x,y);
    printf("swapped:\n");
    swap(x,y);
    printf("x=%d,\ty=%d\n",x,y);
}
```

```
#include <stdio.h>
void swap2(int x,int y)
{   int z;
    z=x;      x=y;      y=z;
}
main()
{   int a[2]={1,2};
    swap2(a[0],a[1]);
    printf("a[0]=%d\na[1]
     =%d\n", a[0],a[1]);
}
```

```
#include <stdio.h>
void swap2( int  x[] )
{   int z;
    z=x[0];     x[0]=x[1];     x[1]=z;
}
main()
{   int a[2]={1,2};
    swap2(a);
    printf("a[0]=%d\na[1]=%d\n",
    a[0],a[1]);
}
```

第11讲：指　针

指针是C语言重要组成部分，精华所在

C程序设计中使用指针可以：

- 使程序简洁、紧凑、高效
- 有效地表示复杂的数据结构
- 动态分配内存
- 得到多于一个的函数返回值

学习指针是学习C语言中最重要的一环，能否正确理解和使用指针是我们是否掌握C语言的一个标志，可以说不懂C语言中的指针就不懂什么是C语言。

学习内容

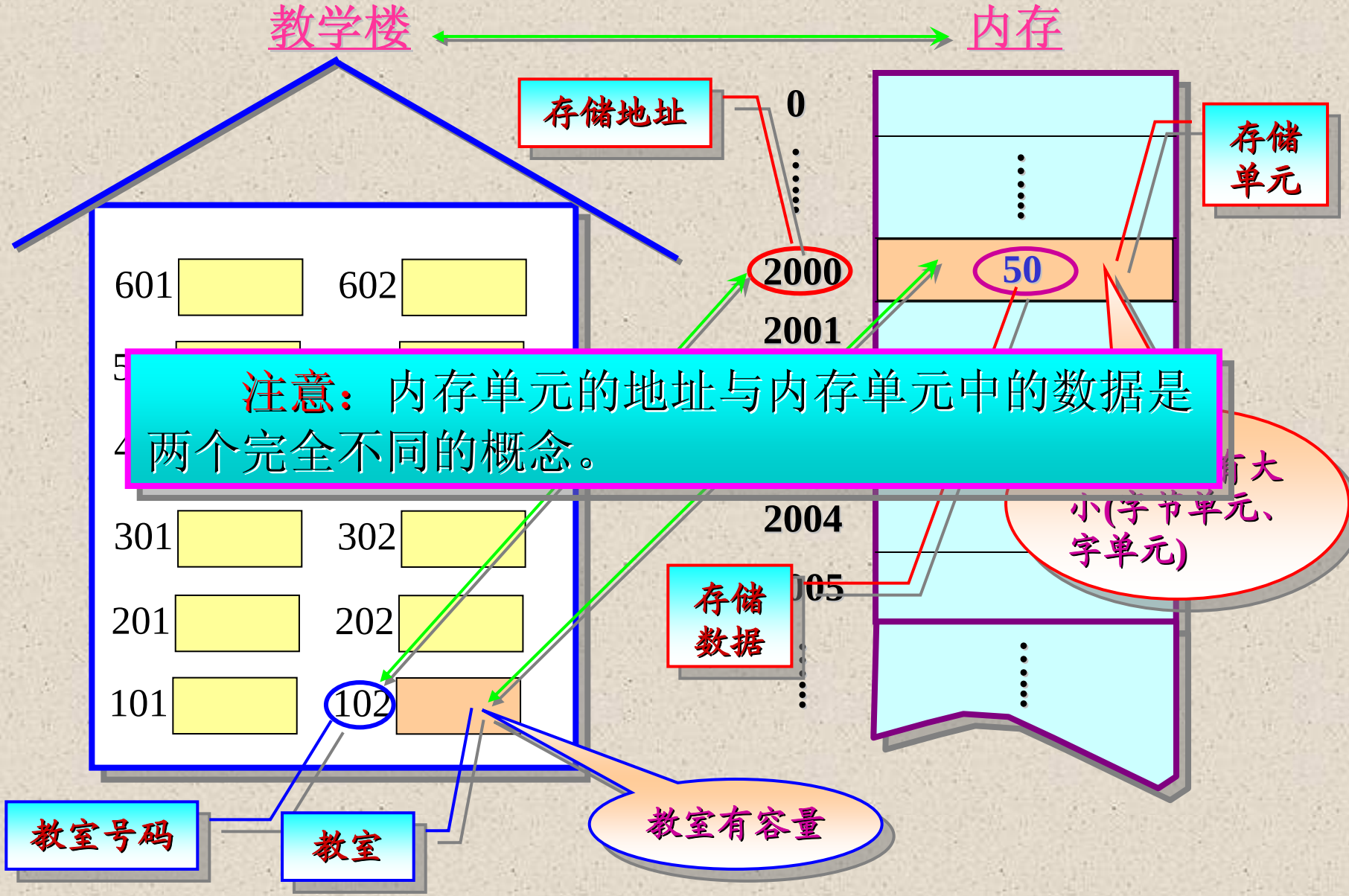
- 指针概念
- 指针运算
- 指针与数组
- 堆分配内存
- 字符指针

学习目的

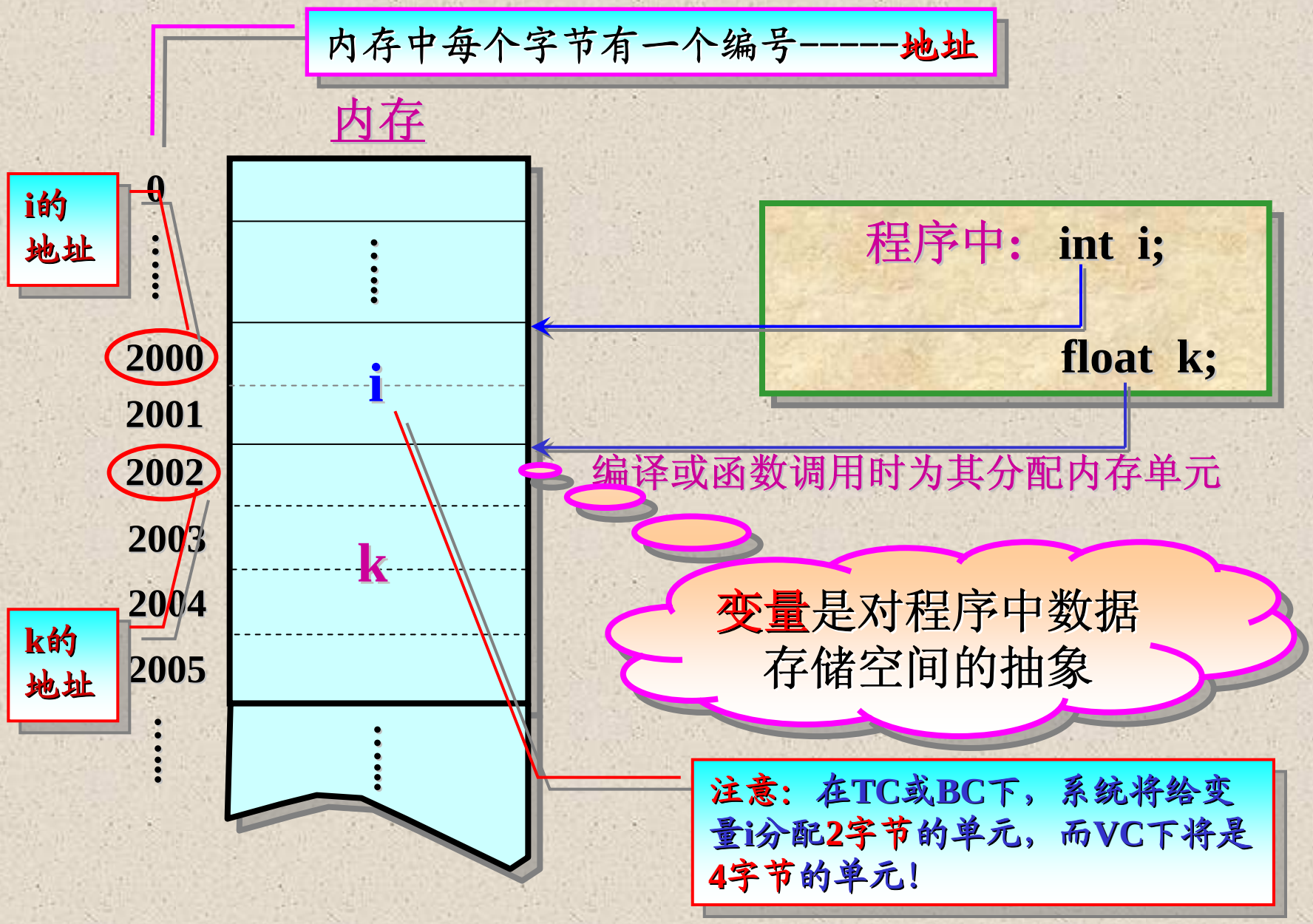
- 理解指针数据类型
- 掌握指针的基本运算
- 使用指针对数组进行操作
- 使用指针动态分配内存
- 使用指针作为函数参数
- 掌握字符指针的使用

8.1 指针与指针变量的概念

1、内存地址——内存中存储单元的编号

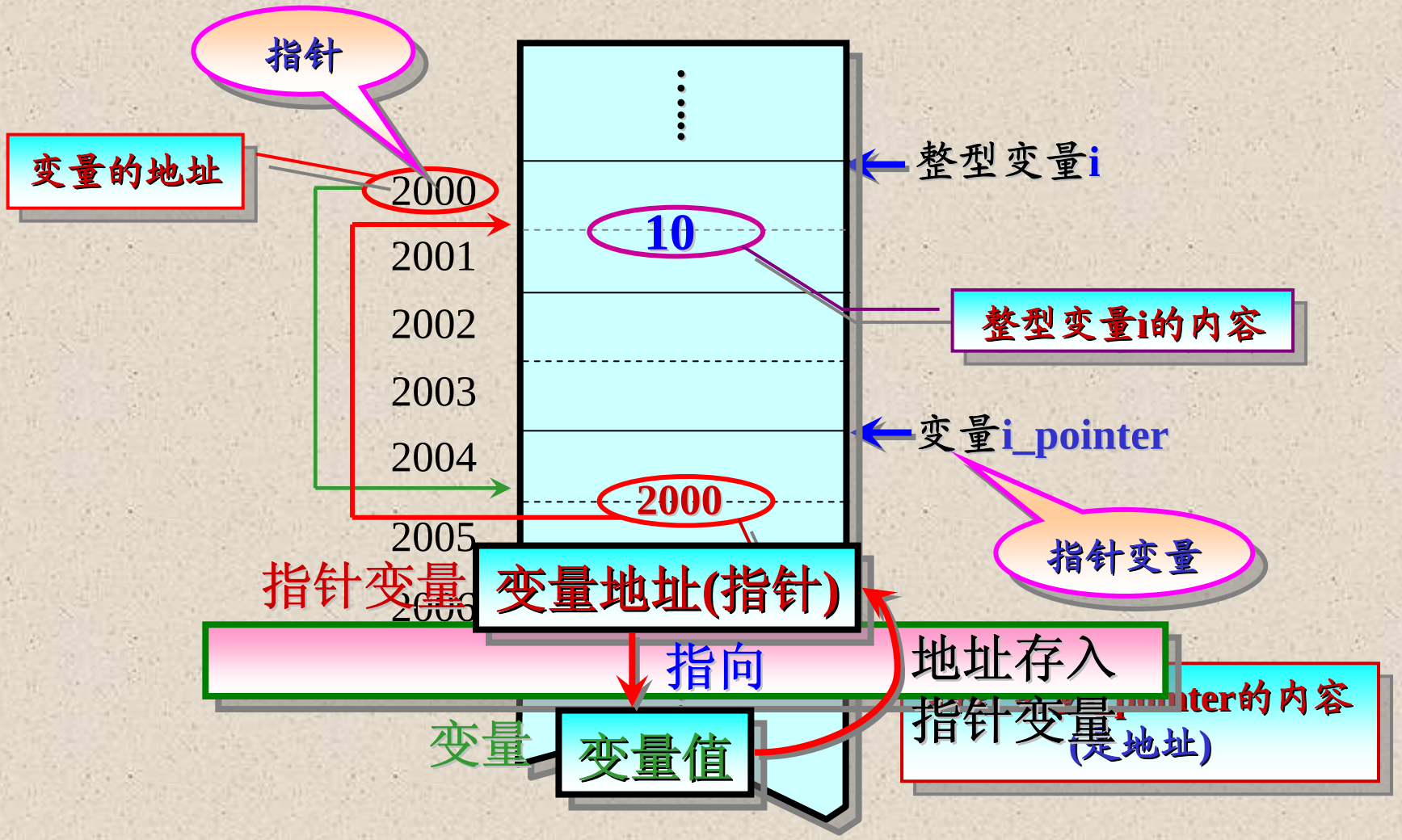


2、变量地址——系统分配给变量的内存单元的起始地址



3、指针与指针变量

- 指针：一个变量的地址
- 指针变量：专门存放变量地址的变量

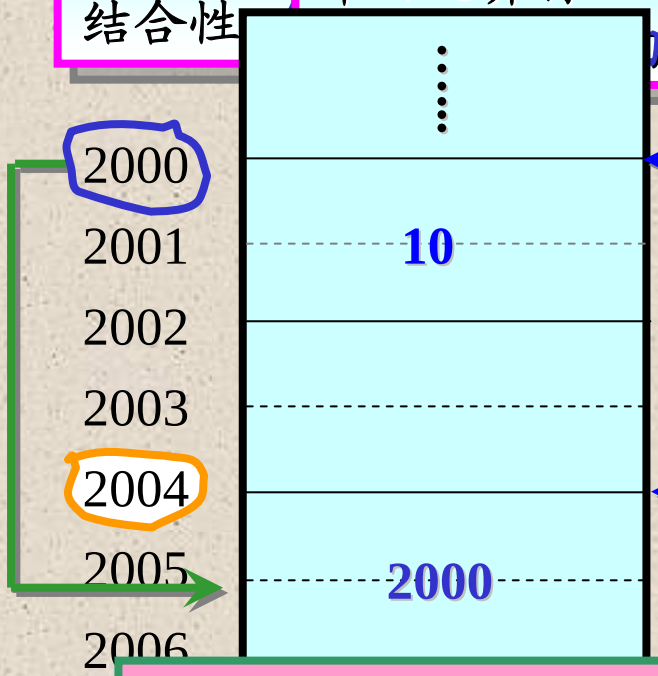


4、&与*运算符

- 含义
- 两者关系：互为逆运算
- 理解

含义：取指针所指向变量的内容
单目运算符
&i_pointer

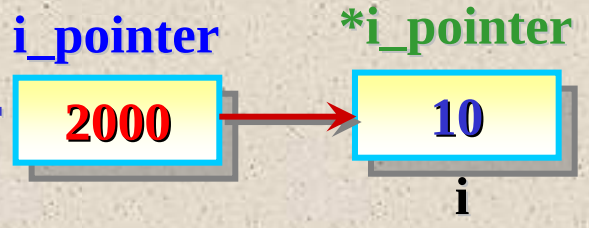
含义：取
单目运算
结合性



整型变量i

变量i_pointer

指针变量



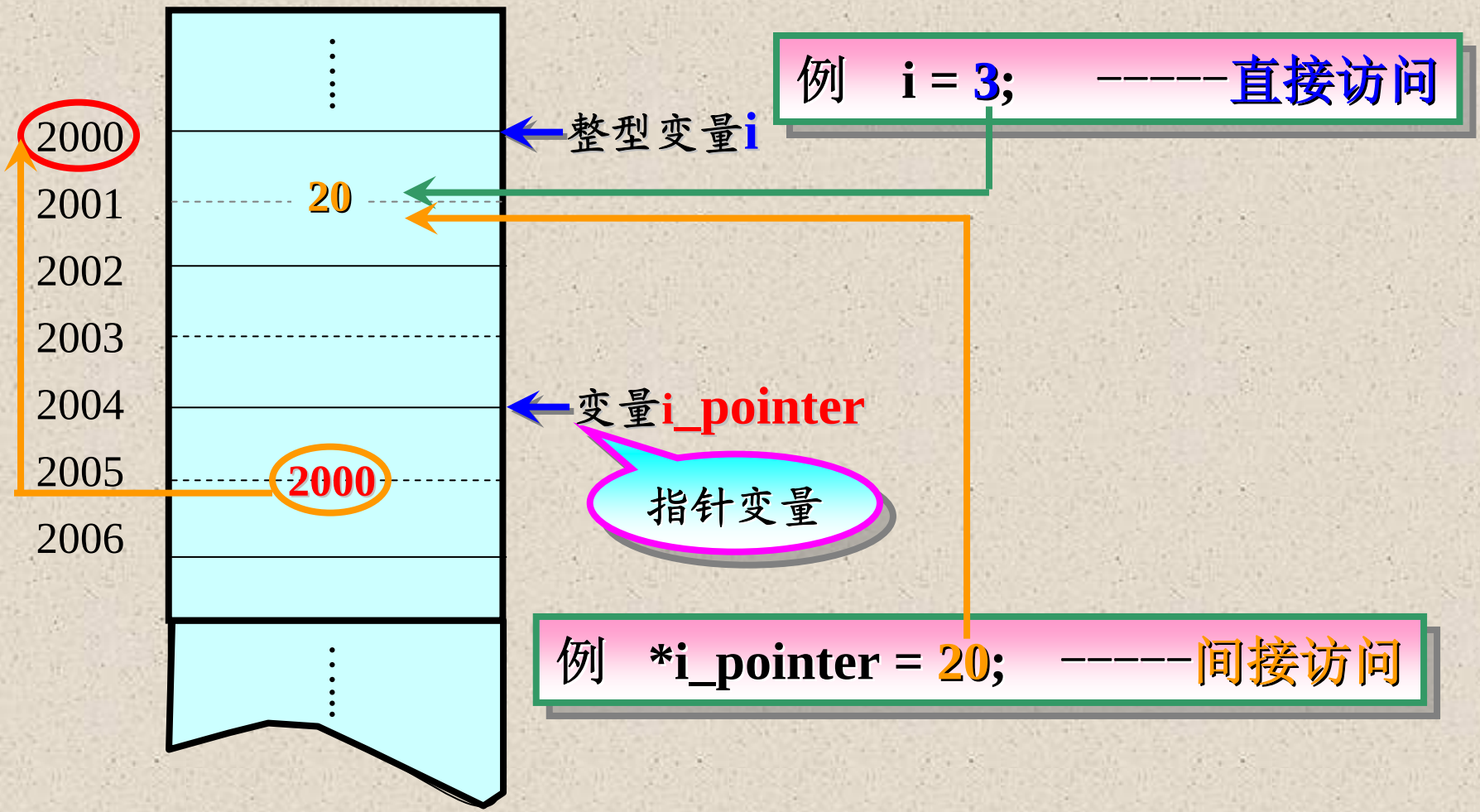
i_pointer = &i = &(*i_pointer)
i = *i_pointer = *(&i)

i_pointer-----指针变量，它的内容是地址量
***i_pointer**----指针的目标变量，它的内容是数据
&i_pointer---指针变量占用内存的地址

8.2 指针变量的定义和引用

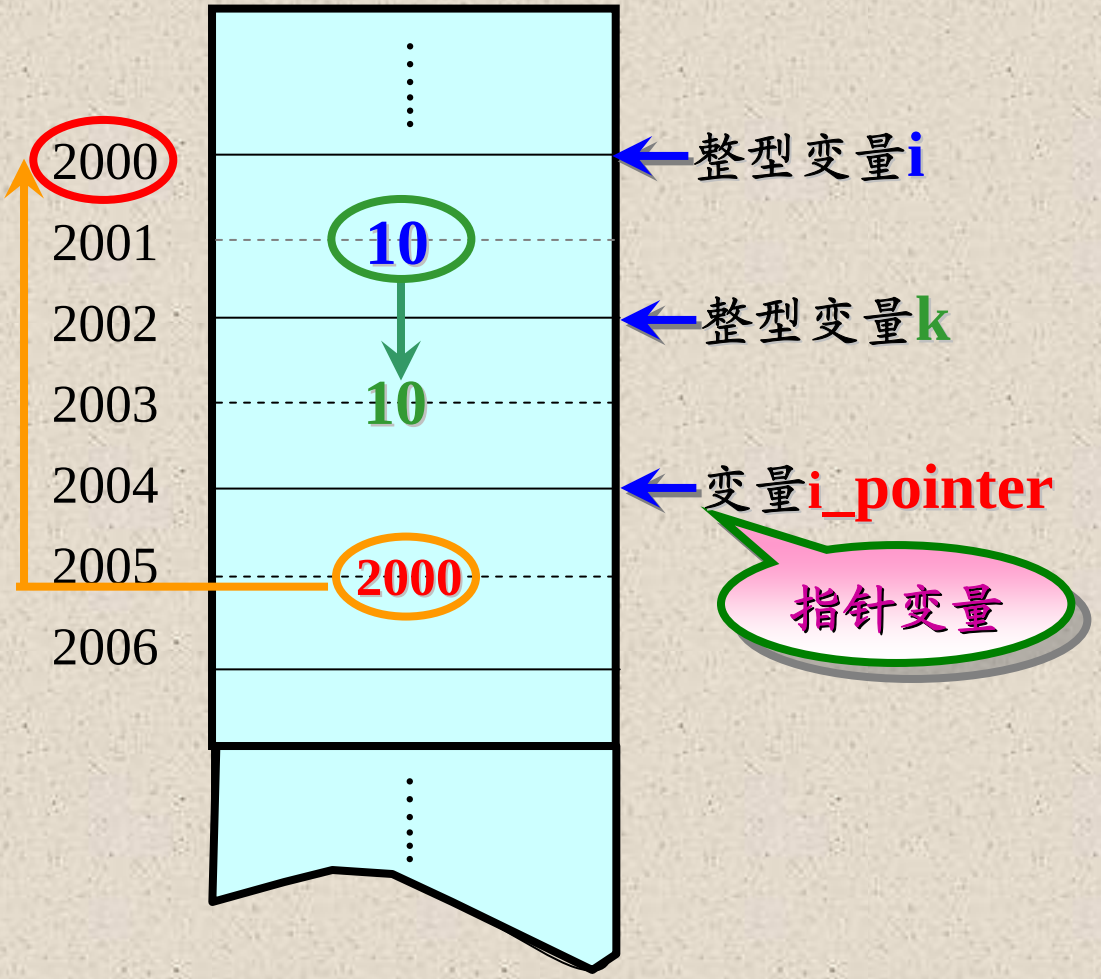
1、变量值的存取方法

- 直接访问：按变量名来存取变量值
- 间接访问：通过存放变量地址的变量去访问变量

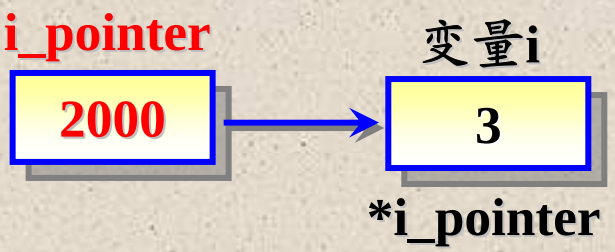


例　k = i;　　--直接访问

k = *i_pointer;　　--间接访问



2、指针变量与其所指向的变量之间的关系



i ⟺ *i_pointer

&i ⟺ i_pointer

i=3; ⟺ *i_pointer=3;

3、指针变量的定义

一般形式：[存储类型] 数据类型符 *变量名;

例
```
int   *p1, *p2;
float *q;
static char *name;
```

标识符

注意：

- int *p1, *p2; 与 int *p1, p2;
- 指针变量名是p1,p2 ,不是*p1,*p2
- 指针变量只能指向定义时所规定类型的变量
- 指针变量定义后，变量值不确定，应用前必须先赋值

4、指针变量的赋值

➢ 初始化赋值

[存储类型] 数据类型 *指针名 = 初始地址值;

例
```
int i;
int *p = &i;
```

赋给指针变量，
不是赋给目标变量

例
```
int i;
int *p = &i;
int *q = p;
```

变量必须已说明过
类型应一致

×

用已初始化指针变量作初值

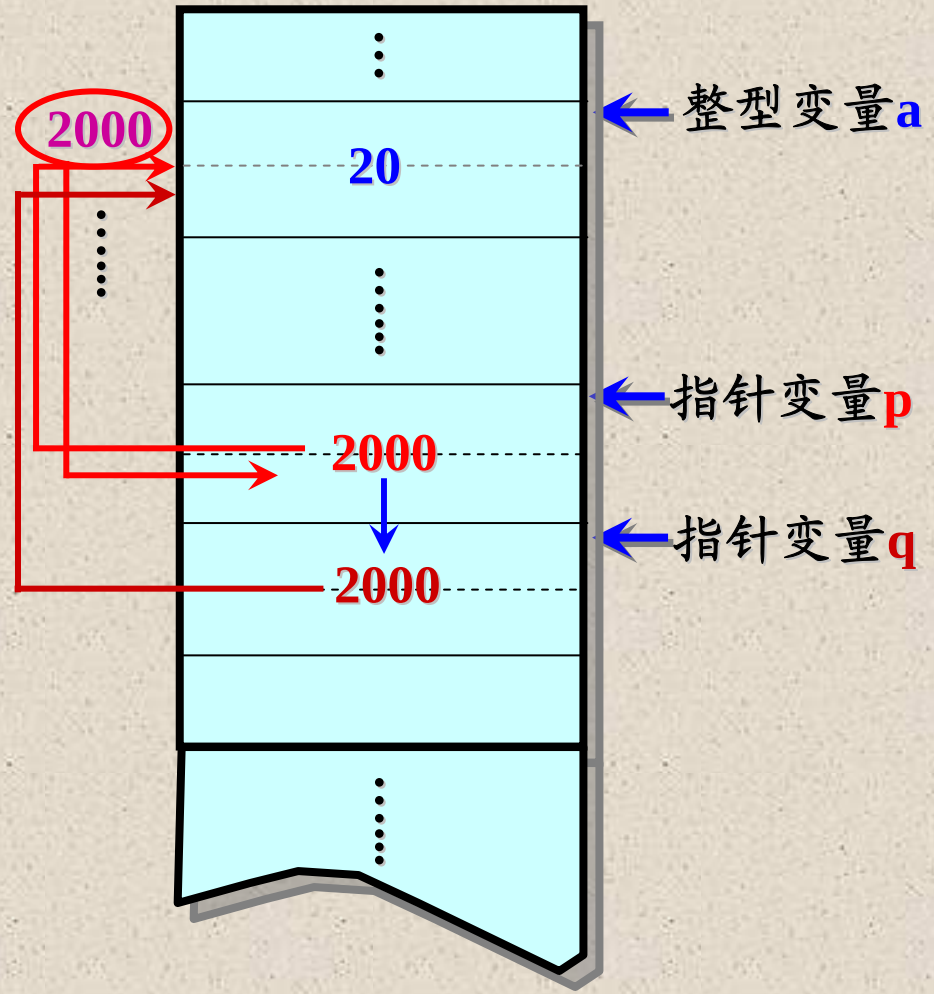
例
```
void main ( )
{
  int i;
  static int *p = &i;
  ..............
}
```
(×)

不能用auto变量的地址
去初始化static型指针

➢ 赋值语句赋值

例
```
int  a;
int  *p;
p = &a;
```

例
```
int  a = 20 ;
int  *p, *q;
p = &a;
q = p;
```



指针变量赋值的几种错误方法:

例
```
int *p = &a;
int a;
```

变量a的定义在后，对a的引用超出了a的作用域

注意：一个指针变量只能指向同类型的变量如果给指针赋值时，=号右边的指针类型与左边的指针类型不同，则需要进行类型强制转换。

```
int  a;
int  *pi;
char *pc;
pi = &a;              //pi指向a
pc = (char *)pi;      //pc也指向了a，即pi和pc的值都是a的地址
```

```
p = 2000;
```

给指针变量

例
```
int a;
static int *p = &a;
```

不能用auto变量的地址去初始化static型指针

5、零指针与空类型指针

➢ 零指针：（空指针）

- 定义：指针变量值为零
- 表示： **int * p = 0;**

p指向地址为0的单元，
系统保证该单元不作它用
表示指针变量值没有意义

```
#define  NULL  0
int  *p = NULL:
```

- p = NULL与未对p赋值不同
- 用途：
  - ✓避免指针变量的非法引用
  - ✓在程序中常作为状态比较

例
```
int  *p;
.....
while (p != NULL)
{   ......
}
```

➢ void *类型指针

- 表示：**void *p;**
- 使用时要进行强制类型转换

表示不
类型数

例
```
char  *p1;
void  *p2;
p1=(char  *)p2;
p2=(void *)p1;
```

6、引用指针变量

格式:　　*指针变量

```
int a;
int *p = &a;     // p指向a
*p = 10;         // 相当于 a = 10;
```

```
int a, *p;
p = &a;
*p = 10;
a++;
printf ("a = %d, *p = %d", a, *p);
```

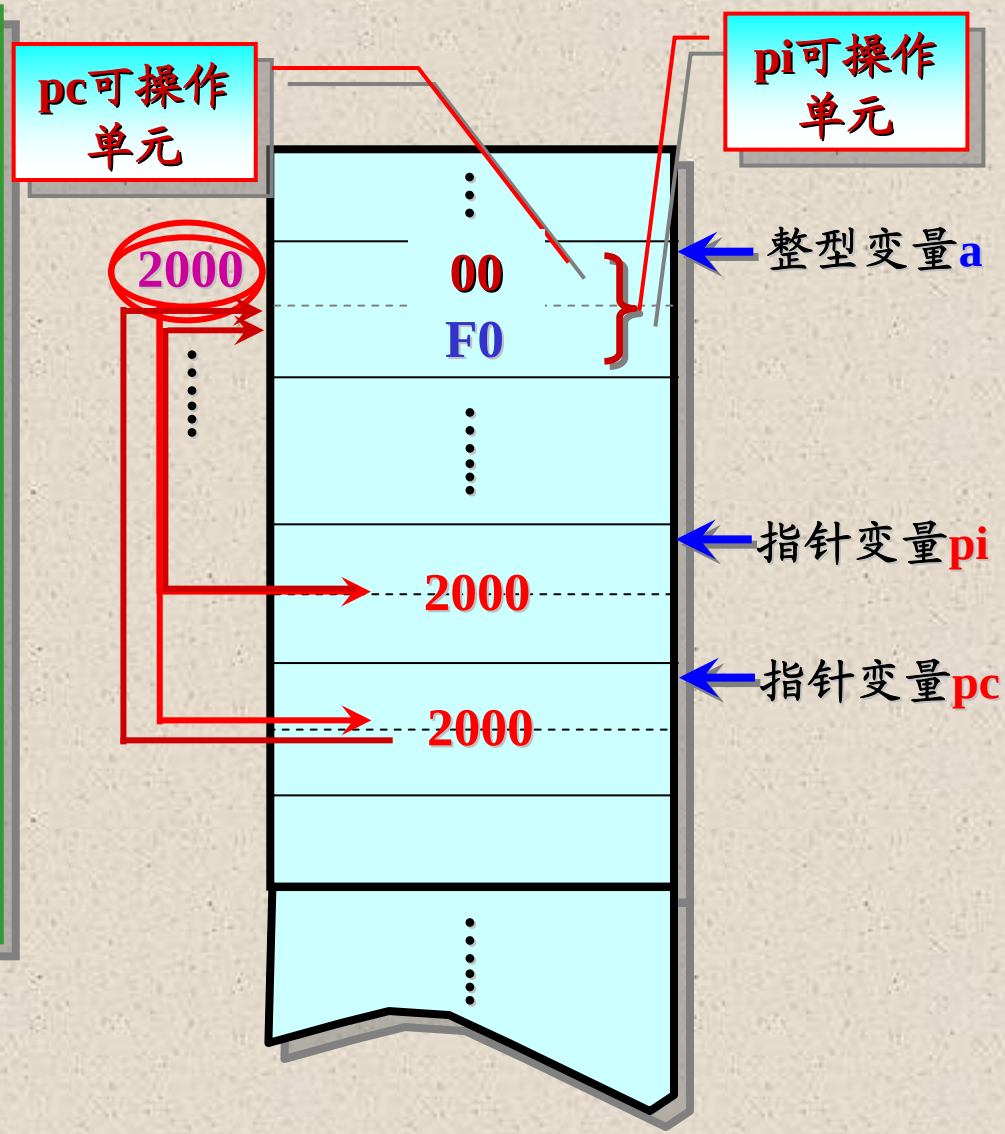
输出结果:

a = 11, *p = 11

注意：程序在利用指针间接引用内存单元时，将按照指针变量定义时所指向的数据类型来解释引用的内存单元。

【例1】不同类型的指针操作同一内存变量

```
#include <stdio.h>
void main ( )
{
   unsigned short a;
   unsigned short *pi = &a;
   char *pc = (char *)&a;
   *pi = 0XF0F0;
   *pc = 0;
   printf ("a = %X", a);
}
```

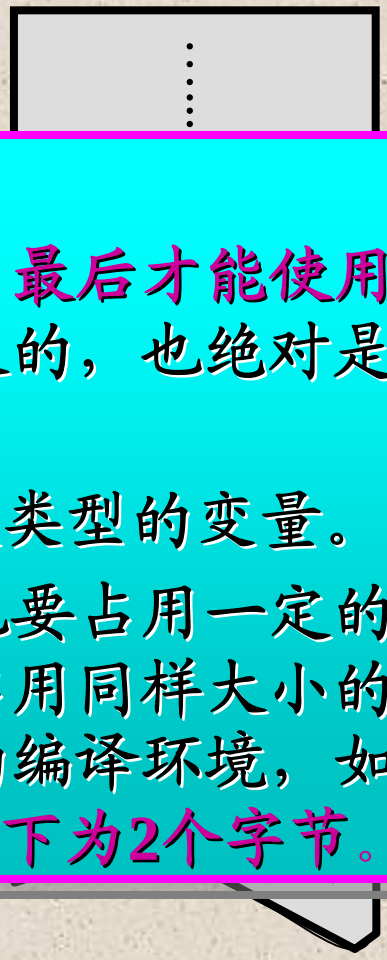


输出结果：

a = F000

【例2】输入两个数，并使其从大到小输出

```
#include <stdio.h>
void  main ( )
{
  i
  s
  I
  i
  {
  I
  I
}
```



重点强调：

- 指针变量必须**先定义，后赋值，最后才能使用**！没有赋值的指针变量是没有任何意义的，也绝对是不允许使用的。
- 指针变量只能指向定义时所规定类型的变量。
- 指针变量也是变量，在内存中也要占用一定的内存单元，但所有类型的指针变量都占用同样大小的内存单元，其具体大小取决于所使用的编译环境，如**在BC3.1和VC6.0下为4个字节，在TC2.0下为2个字节。**

```
a=5, b=9
max=9, min=5
```

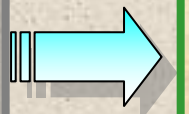
8.3 指针和地址运算

1、指针变量的加、减运算

指针可以参与加法和减法运算，但其加、减的含义绝对不同于一般数值的加减运算。如果指针p是这样定义的：**ptype *p;** ，并且p当前的值是ADDR，那么：

p ± n 的值 = ADDR ± n * sizeof（ptype）

```
int *pi;
char *pc;
long *pl;
pi = (int *) 1000;
pc = (char *) 1000;
```



```
pi++;     //pi的值将是1002 (假设int型占2byte)
pi -= 2;  // pi的值将是998
pc++;     // pc的值将是1001
pc -= 2;  // pc的值将是999
pl++;     // pl的值将是1004
```

注意：两个指针相加没有任何意义，但两个指针相减则有一定的意义，可表示两指针之间所相差的内存单元数或元素的个数，在后面的学习中就会体会到。

2、指针变量的关系运算

- 若p1和p2指向同一数组，则
  - ✓ p1<p2　　表示p1指的元素在前
  - ✓ p1>p2　　表示p1指的元素在后
  - ✓ p1==p2　　表示p1与p2指向同一元素
- 若p1与p2不指向同一数组，比较无意义
- p==NULL或p!=NULL

小结

- 指针的概念
- 指针变量的定义
- 指针变量的类型
- 指针调用——先赋值，后使用
- 指针传递参数的方法及特点

```
int  i,*p;
i=10;
p=&i;
```

提问

1、程序如下：定义一个整型指针变量p，指向x，说出程序运行结果。

```
void fun(int  *p1)
{
     *p1+=10;
}
void main()
{   int  x=15;
    _____;
    _____;
    fun(p);
    printf("x=%d",x);
}
```

int *p;
p=&x;

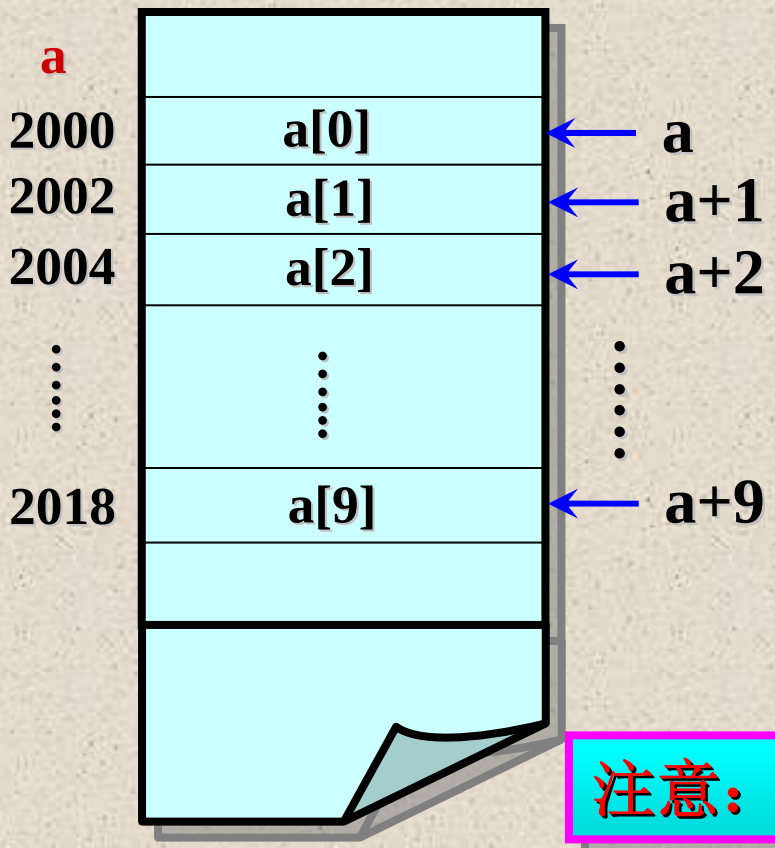
运行结果：
X=25

8.4 指针与数组

1、数组的指针

数组的指针其实就是数组在内存中的起始地址。而数组在内存中的起始地址就是数组变量名，也就是数组第一个元素在内存中的地址。

例：short int a[10];




```
int a[10];
int k;
for (k = 0; k < 10; k++)
   a[k] = k;        //利用数组下标
```

```
int a[10];
int k;
for (k = 0; k < 10; k++)
   *(a+k) = k;  //利用数组的指针
```

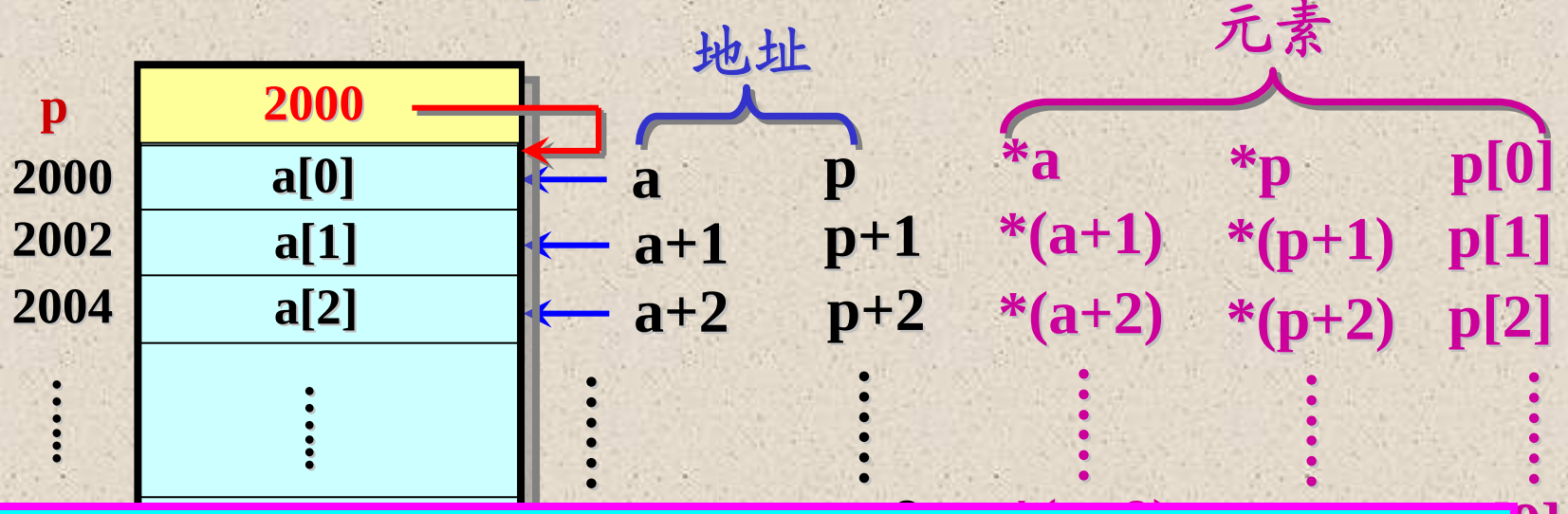
注意：a+k ⟷ &a[k]　　*(a+k) ⟷ a[k]

8.4 指针与数组

2、指向数组的指针变量

如果将数组的起始地址赋给某个指针变量，那么该指针变量就是指向数组的指针变量。

例：short int a[10], p = a;



注意：p + 1指向数组的下一个元素，而不是简单地使指针变量p的值+1。其实际变化为p+1*size(size为一个元素占用的字节数)。例如，假设指针变量p的当前值为2000，则p+1为2000+1*2=2002，而不是2001。

下面是对数组元素赋值的几种方法，它们从功能上是等价的

```
char str[10];
int k;
for (k = 0; k < 10; k++)
   str[k] = 'A' + k;   //也可写成*(str+k) = 'A' + k
```

```
char str[10];
int k;
char *p;
p = str;
for (k = 0; k < 10; k++)
    p[k] = 'A' + k;    //也可写成*(p+k) = 'A' + k
```

执行完后，p仍然指向数组str的首地址

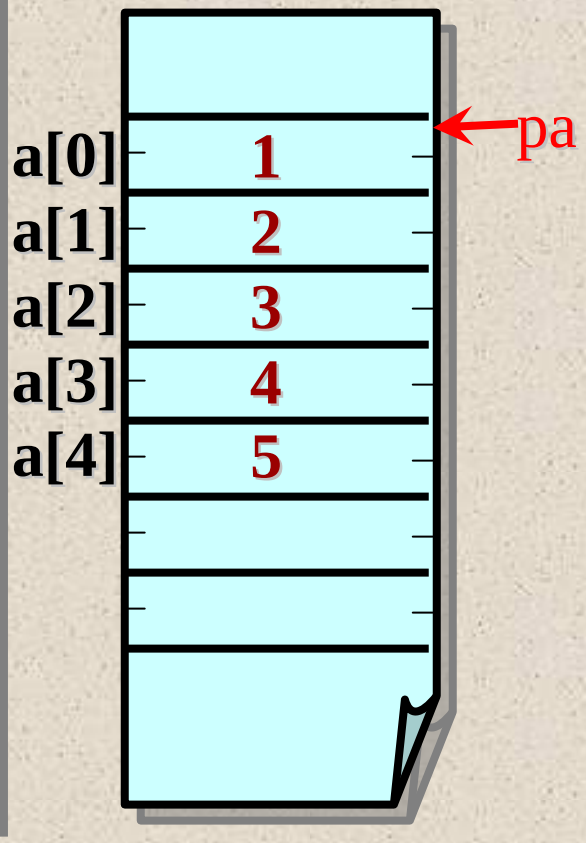
执行完后，p指向数组

注意：数组名是地址常量，切不可对其赋值，也不可做++或--运算。例如：int a[10];如果在程序中出现a++或a--则是错误的。

```
for (k = 0; k < 10; k++)
    *p++ = 'A' + k;  //相当于 *p = 'A' + k; p++;
```

【例】数组元素的引用方法

```
void main( )
{
   int a[5], *pa, i;
   for (i = 0; i < 5; i++)
      a[i]=i+1;
    pa = a;
    for (i = 0; i < 5; i++)
       printf ("*(pa+%d):%d\n", i, *(pa+i));
    for (i = 0; i < 5; i++)
       printf ("*(a+%d):%d\n", i, *(a+i));
    for (i = 0; i < 5; i++)
       printf ("pa[%d]:%d\n", i, pa[i]);
    for (i = 0; i < 5; i++)
       printf("a[%d]:%d\n", i, a[i]);
}
```



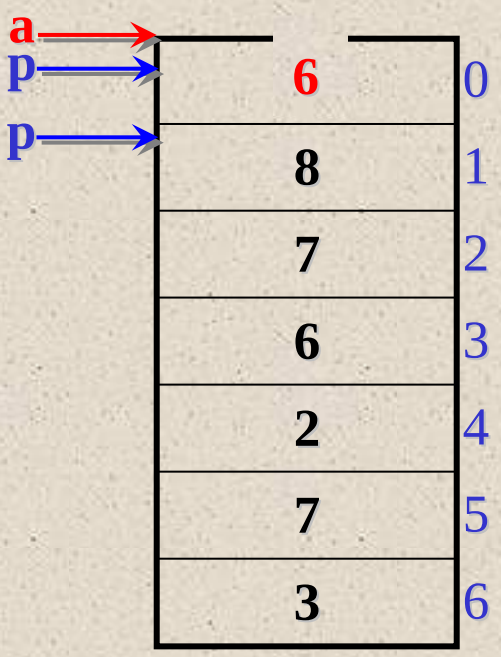
例：int a[ ] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }, *p = a, i;

数组元素地址的正确表示：

（A）&(a+1)　（B）a++　（C）&p　√（D）&p[i]

例：注意指针变量的运算

```
void main()
{
    int  a[ ] = {5, 8, 7, 6, 2, 7, 3};
    int y, *p = &a[1];
    y = (*--p)++;
    printf ("%d  ", y);
    printf ("%d", a[0]);
}
```



输出结果：

5　6

练习

```
prt(int *m,int n)
{
   int i;
   for(i=0;i<n;i++)
     m[i]++;
}
main()
{
   int a[]={1,2,3,4,5},i;
   prt(a,5);
   for(i=0;i<5;i++)
    printf("%d,",a[i]);
}
```

运行结果：
2，3，4，5，6，

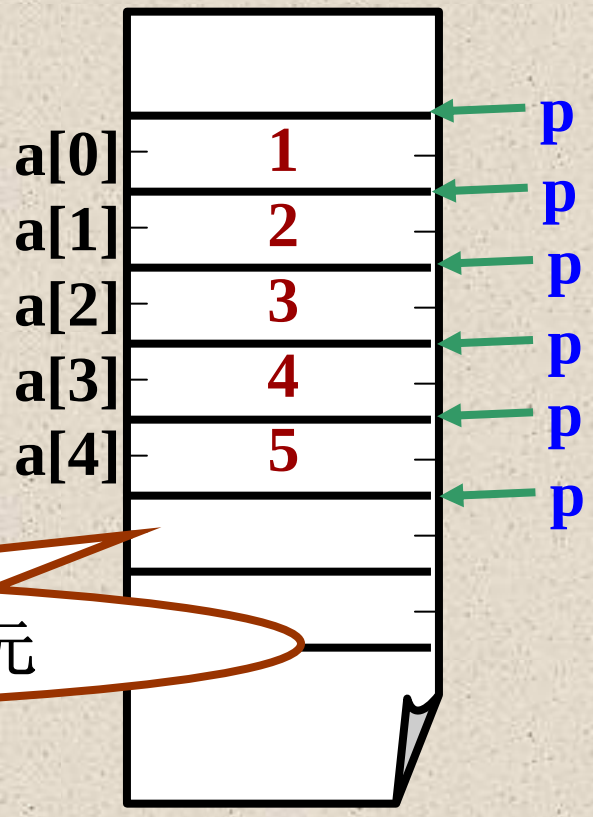
程序运行后的输出结果是什么？

思考

注意指针的当前值

```
main()
{     int a[5],*p=a,i;
      for(i=0;i<5;i++)
            scanf("%d", p++);
      for(i=0;i<5;i++,p++)
            printf("%d: %d\n",i,  *p);
}
```

p=a;



指针变量可以指到数组后的内存单元

能不能将p++改为a++

练习

用选择法对10个整数从大到小排序。（用指针）

exc11_1.cpp

exc11_1_xiugai.cpp

练习

用选择法对10个整数从大到小排序。（用指针）

exc11_1.cpp

```
#include <stdio.h>
void sort(int x[ ],int n);
void main()
{
int *p,i,a[10];
    p=a;
     for(i=0;i<10;i++)
      scanf("%d",p++);
    p=a;
     sort(p,10);
     for(p=a,i=0;i<10;i++)
    {printf("%d ",*p);p++;}
}
```

```
void sort(int x[],int n)
   { int i,j,k,t;
     for(i=0;i<n-1;i++)
     { k=i;
       for(j=i+1;j<n;j++)
       if(x[j]>x[k]) k=j;
       if(k!=i)
       { t=x[i];
         x[i]=x[k];
         x[k]=t;
        }
     }
   }
```

指针与动态内存分配

1、静态内存分配

当程序中定义变量或数组以后，系统就会给变量或数组按照其数据类型及大小来分配相应的内存单元，这种内存分配方式称为静态内存分配 。

```
int k;          //系统将给变量k分配2个字节（VC下分配4个字节）的内存单元
char ch[10];    //系统将给这个数组ch分配10个字节的内存块，首地址就是ch的值
```

静态内存分配一般是在已知道数据量大小的情况下使用

如何解决？
动态内存分配

例如，要对10[illegible]则可定义一个数组：int score[10];[illegible]成绩，然后再进行排序。

```
int n;
int score[n];
scanf ("%d", &n);
```

×

[illegible]学生的具体人[illegible]
人数由用户[illegible]
[illegible]学生的成绩。
[illegible]理呢？

2、动态内存分配

所谓动态内存分配是指在程序运行过程中，根据程序的实际需要来分配一块大小合适的连续的内存单元。

程序可以动态分配一个数组，也可以动态分配其它类型的数据单元。动态分配的内存需要有一个指针变量记录内存的起始地址。

2、动态内存分配

所谓动态内存分配是指在程序运行过程中，根据程序的实际需要来分配一块大小合适的连续的内存单元。

程序可以动态分配一个数组，也可以动态分配其它类型的数据单元。动态分配的内存需要有一个指针变量记录内存的起始地址。

C语言中动态内存分配其实就是使用一个标准的库函数**malloc**，其函数的原型为：

```
void  *malloc( unsigned int size );
```

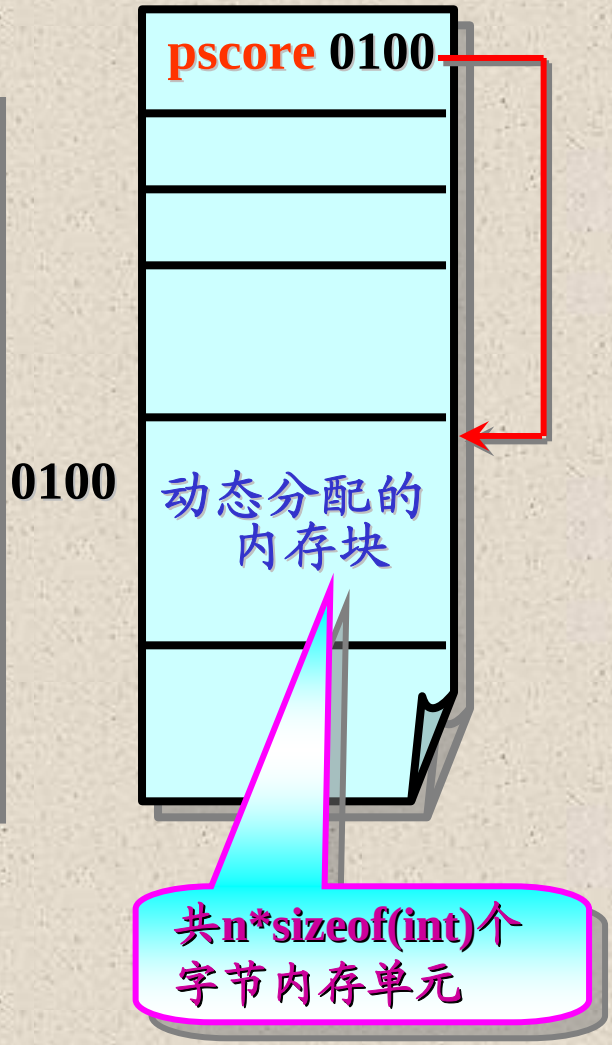
说明：

- **size**这个参数的含义是分配的内存的大小（以字节为单位）。
- 返回值：失败，则返回值是**NULL**（空指针）。
  成功，则返回值是一个指向空类型（**void**）的指针（即所分配内存块的首地址）。

2、动态内存分配

例如：根据学生人数来建立数组的问题可以用动态内存分配来解决，其方法如下：

```
int n, *pscore;
scanf ("%d", &n);
//分配n个连续的整型单元，首地址赋给pscore
pscore = (int *) malloc(n * sizeof(int));
//分配内存失败，则给出错误信息后退出
if (pscore == NULL)
{
  printf ("Insufficient memory available! ");
  exit (0);
}
… …  //可对pscore所指向的单元进行其它处理
```



关于malloc的使用有几点需强调一下：

- malloc前面必须要加上一个指针类型转换符，如前面的(int *)。因为malloc的返回值是空类型的指针，一般应与右边的指针变量类型一致。
- malloc所带的一个参数是指需分配的内存单元字节数，尽管可以直接用数字来表示，但一般写成如下形式：

  分配数量*sizeof(内存单元类型符)

- malloc可能返回NULL，表示分配内存失败，因此一定要检查分配的内存指针是否为空，如果是空指针，则不能引用这个指针，否则会造成系统崩溃。所以在动态内存分配的语句的后面一般紧跟一条if语句以判断分配是否成功

注意：

调用malloc和free函数的源程序中要包含stdlib.h或malloc.h或alloc.h（在TC、BC下）。malloc和free一般成对出现！

的

释放动态内存的函数free其原型为：

```
void  free (void *block);
```

例：　free(pscore);

【例】 编写程序先输入学生人数，然后输入学生成绩，最后输出学生的平均成绩、

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
void main ( )        // exc11
{
  int num, i;
  int maxscore, minscore,
  int *pscore;
  float averscor
  printf ("input
  scanf ("%d",
  if (num <= 0 )
     return;
  pscore = (int *
  if ( pscore ==
  {
    printf ( "Insu
    exit (0);
  }
```

```
printf ("input the scores of students now:\n");
for (i = 0; i < num; i++)
    scanf ("%d", pscore + i);
 maxscore = pscore[0];
 minscore = pscore[0];
 sumscore = pscore[0];
 for (i = 1; i < num; i++)
 {
   if (pscore[i] > maxscore)
```

```
------\n");
ts is %.1f\n",

s is %d\n",

is %d\n",
```

```
  free (pscore);  //释放动态分配的内存
}
```

运行结果:

input the number of student：4↙

input the scores of students now：

45 76 88 94↙

---

the average score of the students is 75.8

the highest score of the students is 94

the lowest score of the students is 45

8.4 堆内存分配

- 堆内存：存放程序动态数据的内存空间，允许程序根据需要申请一定大小的空间，使用完毕可以释放该空间。
- 堆内存的分配关键字：**new**
- 堆内存的释放关键字：**delete**

分配一个数据类型空间

分配:　　数据类型 *　指针名=new 数据类型;

释放:　　delete　指针名;

例如:　int *p;

*p=1;

或:　int *p=new int(1);

delete p;

分配多个连续空间:

分配：数据类型 * 指针名=new 数据类型[下标表达式];

```
int* a=new int[10];
```

使用堆内存：a[i]

释放动态分配的空间：delete [ ] a;

new和delete运算符常用于实现数组的动态分配

```
#include<iostream>                                    //exc11_3.cpp
using namespace std;
void main()
{   int arraysize;
    int *array;
    cout<<"please enter the size:\n";
    cin>>arraysize;
    array=new int[arraysize];                        //分配堆内存
    for(int i=0;i<arraysize;i++){
          array[i]=i*2;
          cout<<array[i]<<" ";
    }
    cout<<endl;
    delete[] array;                                  //释放堆内存
}
```

8.5 const 指针

1. 指向常量的指针(常量指针)
2. 指针常量
3. 指向常量的指针常量(常量指针常量)

指向常量的指针

- 常量指针——表示指针指向的对象是常量，而指针本身是变量。常量指针作为函数的形参时，限定该指针指向的对象在函数内不可更改。
- 定义格式:

const <类型说明符> * <指针名> = &<常量名>;

如： **const int a =89,b=60;**

const int *p = &a;

p=&b; //正确

***p=100;** //错误

指针常量

- 指针常量——表示指针本身是常量，而指针指向的对象可以改变。
- 定义格式如下:

<类型说明符> * const <指针名> = &<变量名>;

如:

```
int a =89,b=60;
int *const   p = &a;
p=&b;                //错误
*p=100;    //正确
```

指向常量的指针常量

- 指向常量的指针常量——表示指针指向的对象和指针本身都是常量。
- 定义格式如下:

const <类型说明符> *const <指针名>=&<常量名>;

如: **const int a =89,b;**

const int *const p = &a;

p=&b; //错误

***p=100;** //错误

8.6 指针与函数

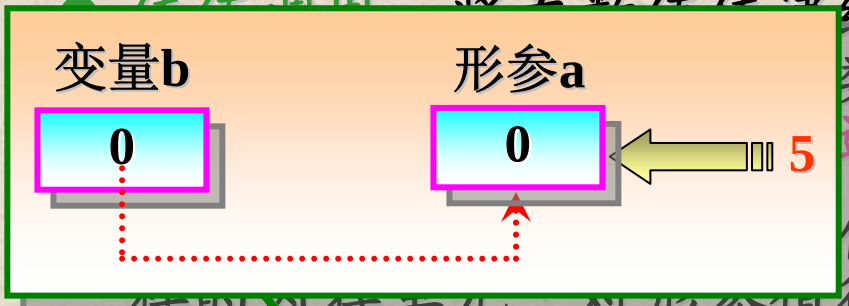
- 数组作函数参数的指针性质：当将数组作为参数传递到函数中，则在栈上定义了指针，可对其进行递增递减的操作。

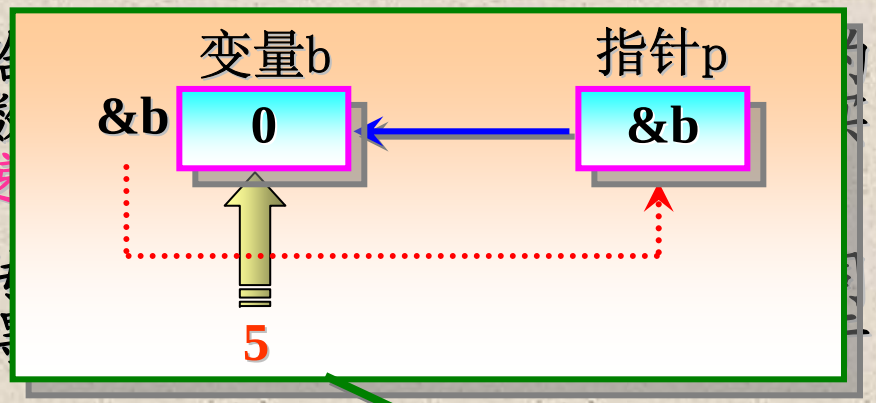
```
#include<iostream>          //ch8_11.cpp
using namespace std;
int sum(int array[],int n);
void main()
{   int a[10]={1,2,3,4,5,6,7,8,9,10};
    cout<<sum(a,10)<<endl;
}
```

```
int sum(int array[],int n)
{       int s=0;
        for(int i=0;i<n;i++)
        { s+=*array;
array++;}
        return s;
}
```

指针作为函数的参数

- 参数传递方式：传值调用和传址调用





双向数据传递方式。

```
void func (int a)
{
   a = 5;
}
void main ( )
{
  int b = 0;
  func (b);
  printf ("b = %d\n", b);
}
```

运行结果：b = 0

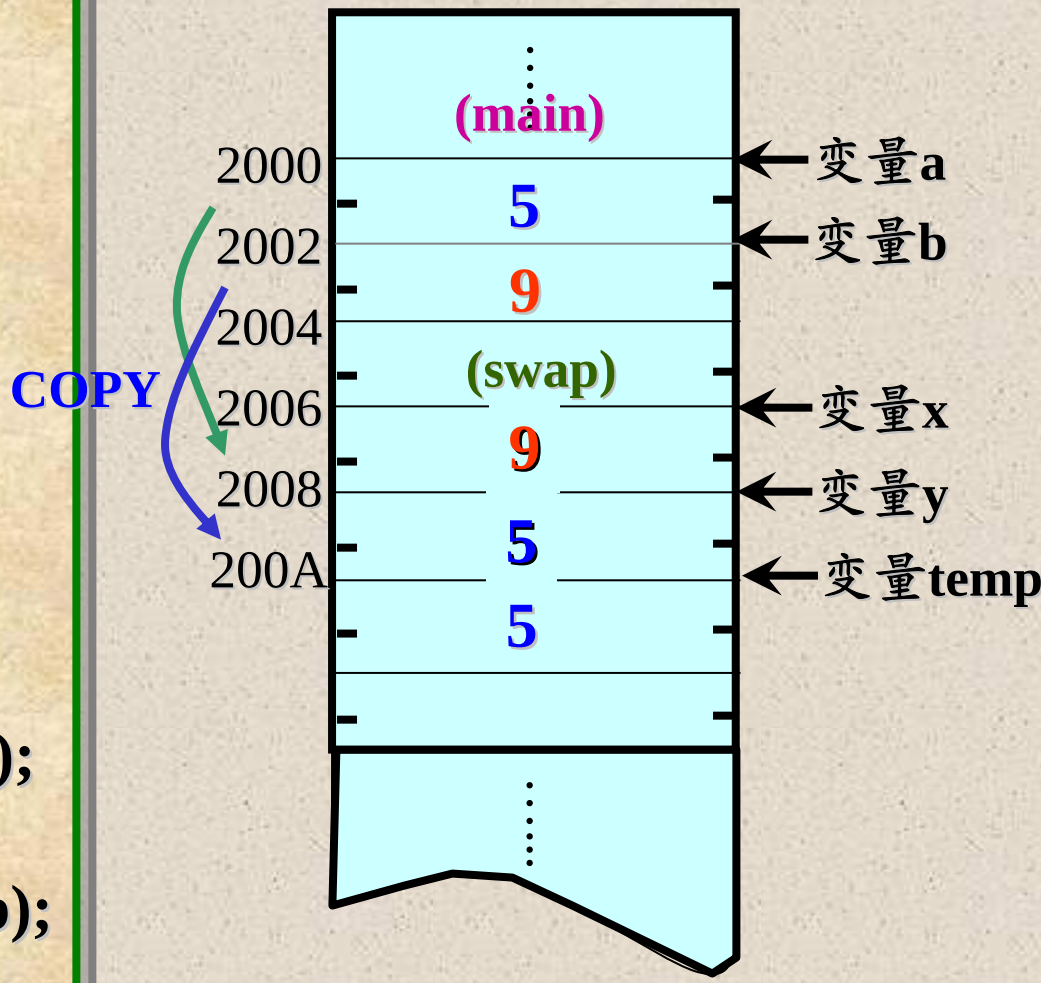
为什么结果不一样呢？

```
void func (int *p)
{
   *p = 5;
}
void main ( )
{
  int b = 0;
  func (&b);
  printf ("b = %d\n", b);
}
```

运行结果：b = 5

【例】将数从大到小输出

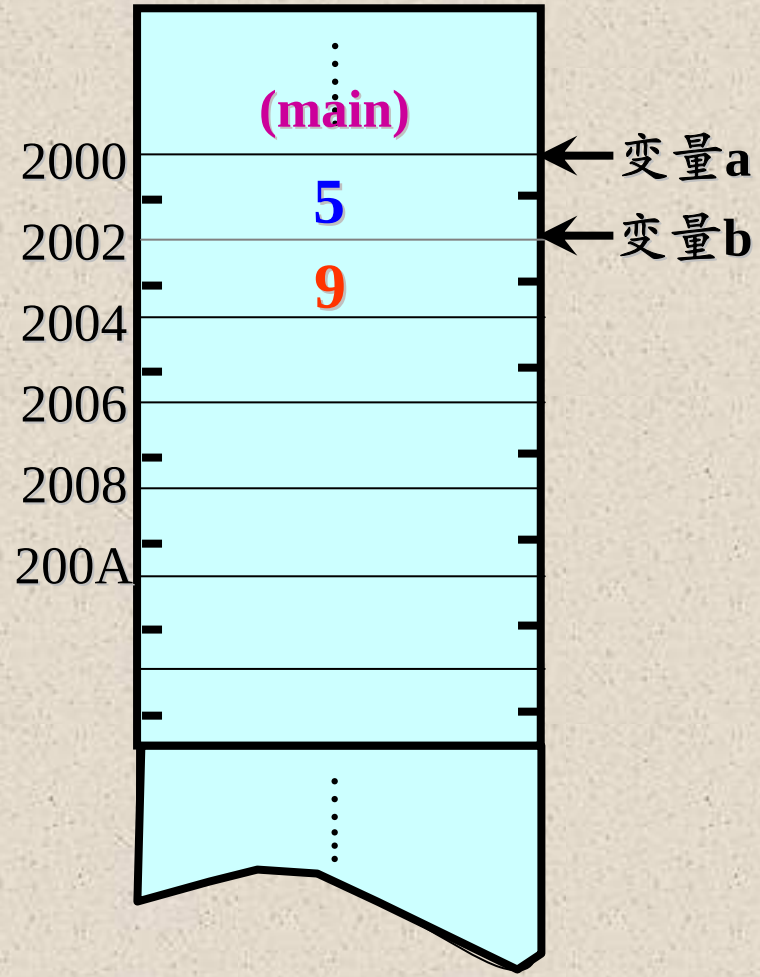
```
void swap (int  x, int y)
{
   int  temp;
   temp = x;
   x = y;
   y = temp;
}
void main ( )
{
   int a, b;
   scanf("%d,%d",&a,&b);
   if(a<b)  swap(a,b);
   printf("\n%d,%d\n",a,b);
}
```



【例】将数从大到小输出

```
void swap (int  x, int y)
{
   int  temp;
   temp = x;
   x = y;
   y = temp;
}
void main ( )
{
   int a, b;
   scanf("%d,%d",&a,&b);
   if(a<b)  swap(a,b);
   printf("\n%d,%d\n",a,b);
}
```

值传递

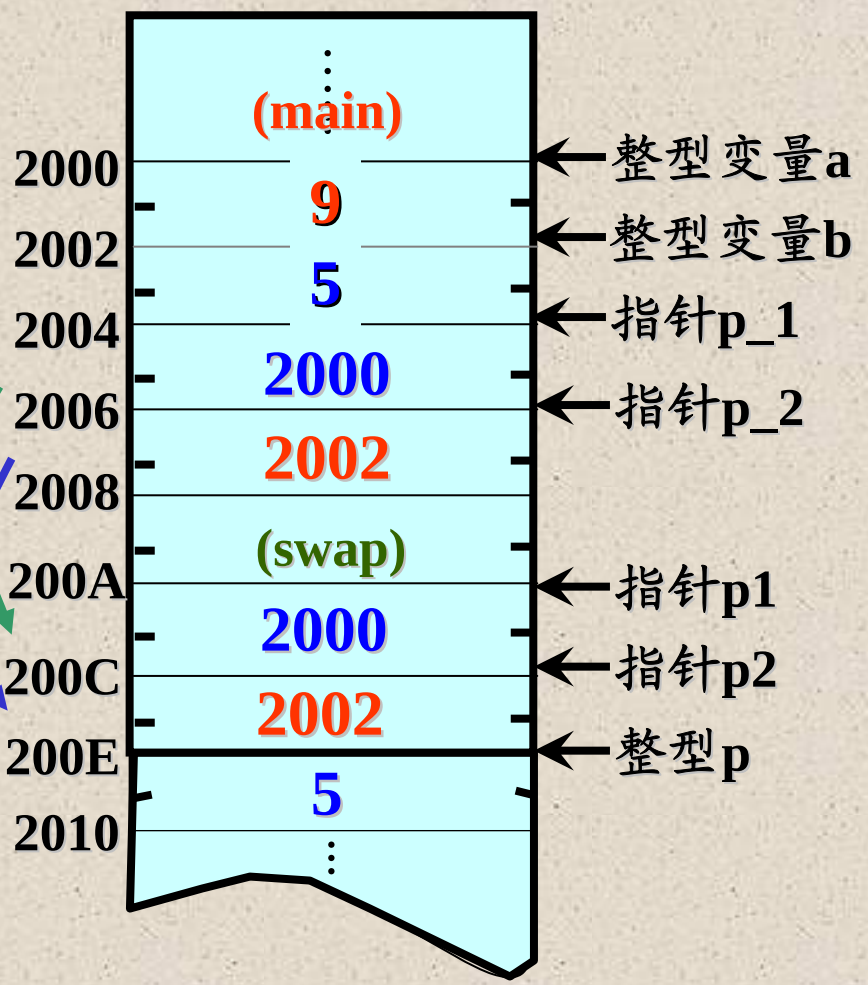


运行结果：5, 9

【例】将数从大到小输出

```
void swap(int  *p1, int  *p2)
{
   int p;
   p = *p1;
   *p1 = *p2;
   *p2 = p;
}
void main ( )
{
    int a, b;
    int *p_1, *p_2;
    scanf ("%d,%d", &a, &b);
    p_1 = &a;  p_2 = &b;
    if (a < b)  swap (p_1, p_2);
    printf ("\n%d,%d\n", a, b);
}
```

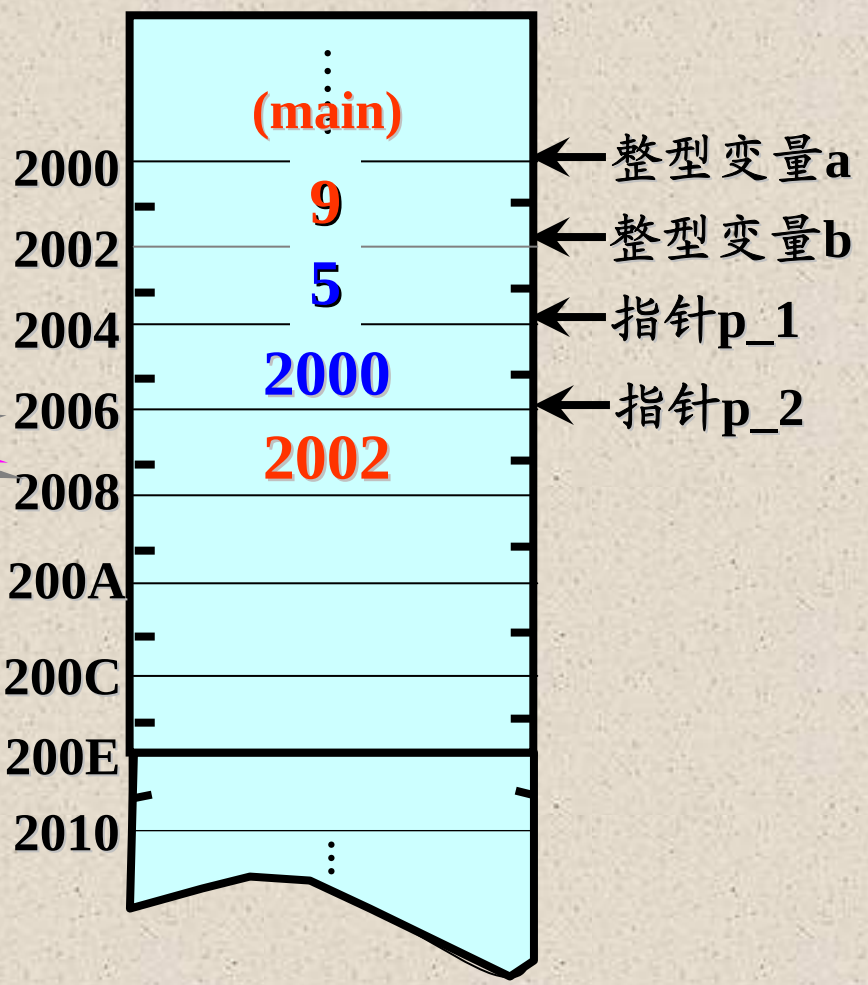
COPY



【例】将数从大到小输出

```
void swap(int  *p1, int  *p2)
{
   int p;
   p = *p1;
   *p1 = *p2;
   *p2 = p;
}
void main ( )
{
    int a, b;
    int *p_1, *p_2;
    scanf ("%d,%d", &a, &b);
    p_1 = &a;  p_2 = &b;
    if (a < b)  swap (p_1, p_2);
    printf ("\n%d,%d\n", a, b);
}
```

地址传递



运行结果：9, 5

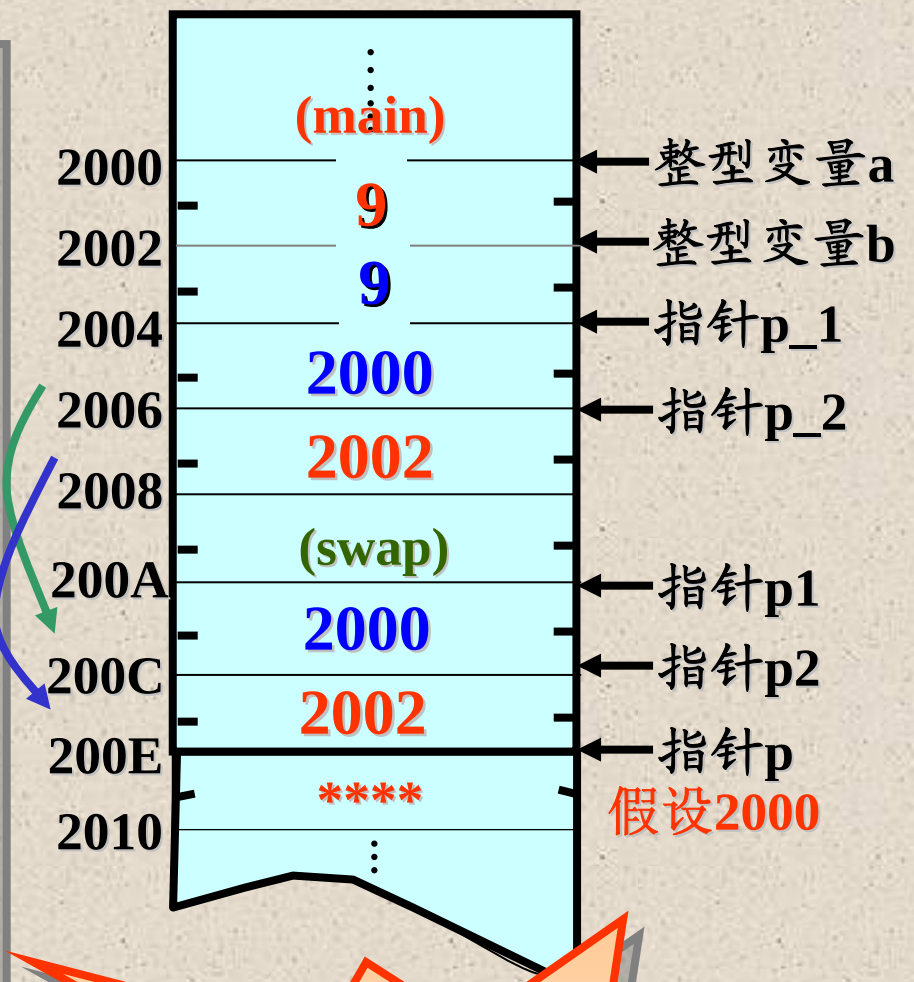
【例】将数从大到小输出

```
void swap (int *p1, int *p2)
{
   int *p;
   *p = *p1;
   *p1 = *p2;
   *p2 = *p;
}
void main ( )
{
   int a,b;
   int *p_1, *p_2;
    scanf ("%d,%d", &a, &b);
    p_1 = &a;  p_2 = &b;
    if (a < b)  swap (p_1, p_2);
    printf ("\n%d,%d\n", a, b);
}
```

int x;
int *p = &x;

COPY

编译警告！
结果不对！



指针变量在使用前
必须赋值！

运行结果：9，9

指针变量作为函数参数

1）地址传递

2）指针变量所指向的变量的值发生变化，函数调用结束后，值保留下来。

3）可以改变主调函数多个变量的值

练习

利用指针及函数调用方法，输入三个整数，按由大到小的顺序输出。

```
swap(int *p1,int *p2)
{   int p;
    p=*p1;
    *p1=*p2;
    *p2=p;
}
```

```
main()
{   int a,b,c;
    int *pt1,*pt2, *pt3;
    scanf("%d,%d, %d ",&a,&b,&c);
    pt1=&a;
    pt2=&b;
    pt3=&c;
    exchange(pt1,pt2,pt3);
    printf("\n%d,%d,%d\n",a,b,c);
}
```

```
exchange(int *q1,int *q2,int *q3)
{ if(*q1<*q2) swap(q1,q2);
  if(*q1<*q3) swap(q1,q3);
  if(*q2<*q3) swap(q2,q3);
}
```

练习

用选择法对10个整数从大到小排序。（用指针）

exc11_1.cpp

exc11_1_xiugai.cpp

练习

用选择法对10个整数从大到小排序。（用指针）

exc11_1.cpp

```
#include <stdio.h>
void sort(int x[ ],int n);
void main()
{
int *p,i,a[10];
    p=a;
     for(i=0;i<10;i++)
      scanf("%d",p++);
    p=a;
     sort(p,10);
     for(p=a,i=0;i<10;i++)
    {printf("%d ",*p);p++;}
}
```

```
void sort(int x[],int n)
   { int i,j,k,t;
     for(i=0;i<n-1;i++)
     { k=i;
       for(j=i+1;j<n;j++)
       if(x[j]>x[k]) k=j;
       if(k!=i)
       { t=x[i];
         x[i]=x[k];
         x[k]=t;
        }
     }
   }
```

练习

用选择法对10个整数从大到小排序。（用指针）

```
//exc11_1_xiugai.cpp

#include <stdio.h>
void sort(int x[ ],int n);
void main()
{
int *p,i,a[10];
    p=a;
     for(i=0;i<10;i++)
      scanf("%d",p++);
    p=a;
     sort(p,10);
     for(p=a,i=0;i<10;i++)
    {printf("%d ",*p);p++;}
}
```

```
void sort(int *x,int n)
    {int i,j,k,t;
      for(i=0;i<n-1;i++)
      {k=i;
       for(j=i+1;j<n;j++)
       if(*(x+j)>*(x+k))
          k=j;
       if(k!=i)
       {t=*(x+i);
        *(x+i)=*(x+k);
        *(x+k)=t;}
      }
    }
```

指针作为函数的返回值____指针函数

一个函数可以返回一个int型、float型、char型的数据，也可以返回一个指针类型的数据。 返回指针值的函数（简称指针函数）的定义格式如下:

```
函数类型  *函数名( [形参1, 形参2,…, 形参n] )
```

指针函数

- 返回指针的函数称为指针函数:

```
数据类型 *函数名（参数列表）
  {
      数据类型  *p;
      ....
      return(p);
  }
```

注意：返回的指针可以是堆地址、全局或静态变量的地址，但不能是局部指针变量。

注意：如果一个函数返回一个指针，不能返回auto型局部变量的地址，但可以返回static型变量的地址。

```
int *getdata (int num)
{
  int a[100];
  int k;

  if (num > 100)
     return (NULL);
  for (k = 0; k < num; k++)
     scanf ("%d", &a[k]);
  return (a);
}
```

错误！

函数返回后数组a将自动释放

```
int *getdata (int num)
{
  static int a[100];
  int k;

  if (num > 100)
     return (NULL);
  for (k = 0; k < num; k++)
     scanf ("%d", &a[k]);
  return (a);
}
```

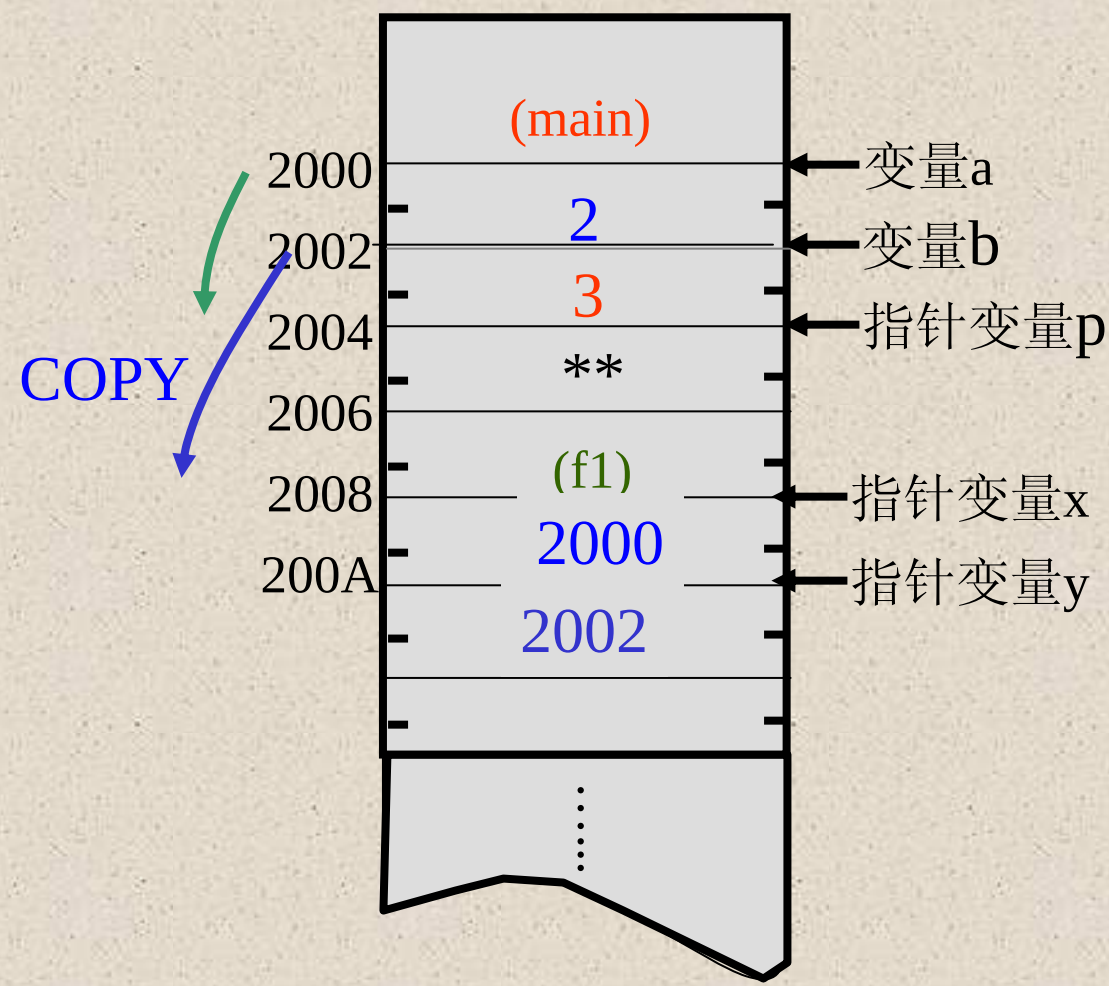
正确！

函数返回后数组a仍将存在

例 写一个函数，求两个int型变量中居于较大值的变量的地址（1）

```
main()
{   int a=2,b=3;
    int *p;
    p=f1(&a, &b);
    printf("%d\n",*p);
}
```

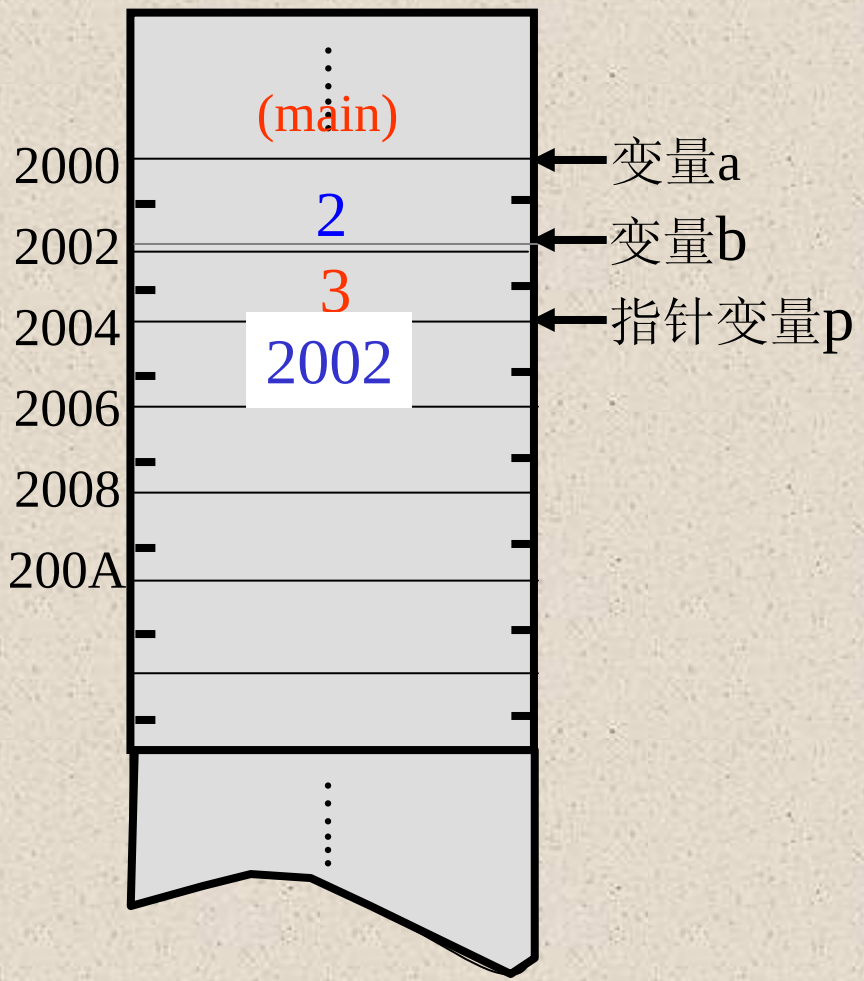
```
int *f1(int *x,int *y)
{
    if(*x>*y)
            return  x;
    else
            return  y;
}
```



例 写一个函数，求两个int型变量中居于较大值的变量的地址（2）

```
main()
{   int a=2,b=3;
    int *p;
    p=f1(&a,&b);
    printf("%d\n",*p);
}
```

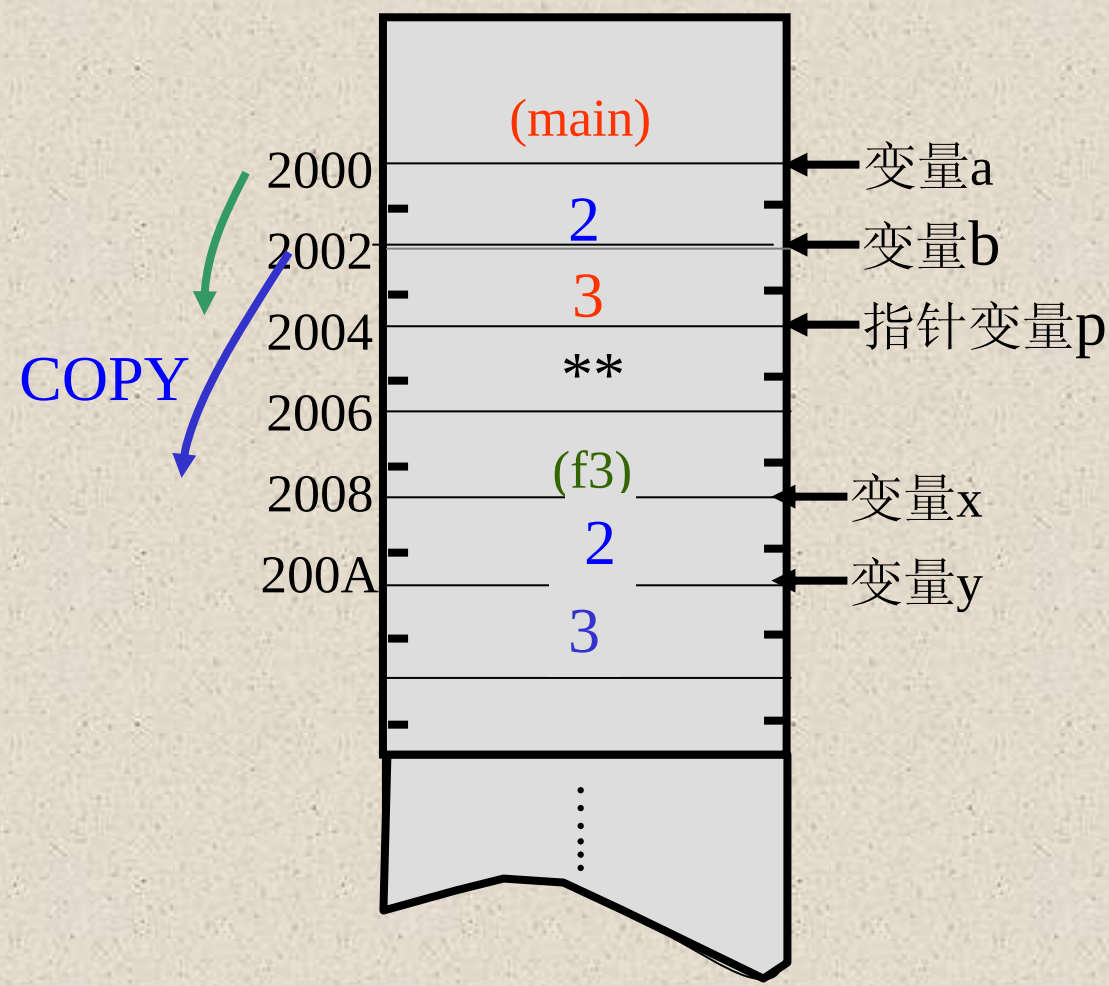
```
int *f1(int *x,int *y)
{
    if(*x>*y)
            return x;
    else
            return y;
}
```



例 写一个函数，求两个int型变量中居于较大值的变量的地址（3）

```
main()
{   int a=2,b=3;
    int *p;
    p=f3(a, b);
    printf("%d\n",*p);
}
```

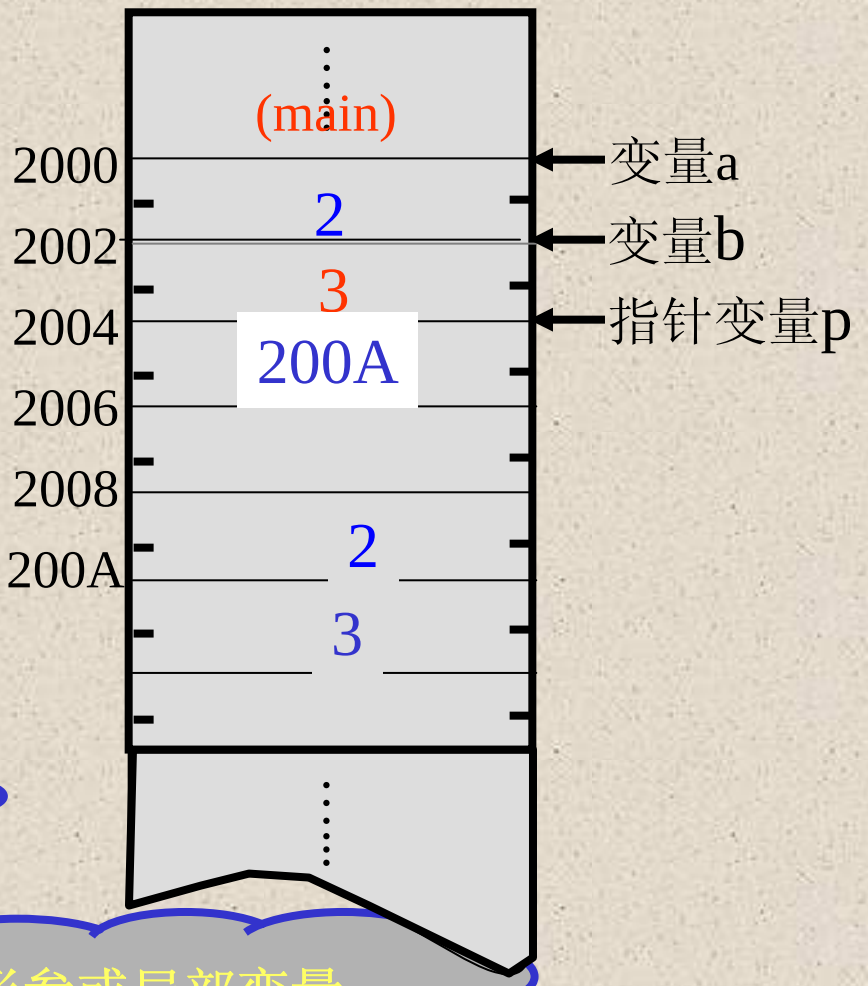
```
int *f3(int x,int y)
{
    if(x>y)
            return  &x;
    else
            return  &y;
}
```



例 写一个函数，求两个int型变量中居于较大值的变量的地址（4）

```
main()
{   int a=2,b=3;
    int *p;
    p=f3(a,b);
    printf("%d\n",*p);
}
```

```
int *f3(int x,int y)
{
    if(x>y)
            return &x;
    else
            return &y;
}
```



不能返回形参或局部变量的地址作函数返回值

练习

- 利用指针编写一个子函数

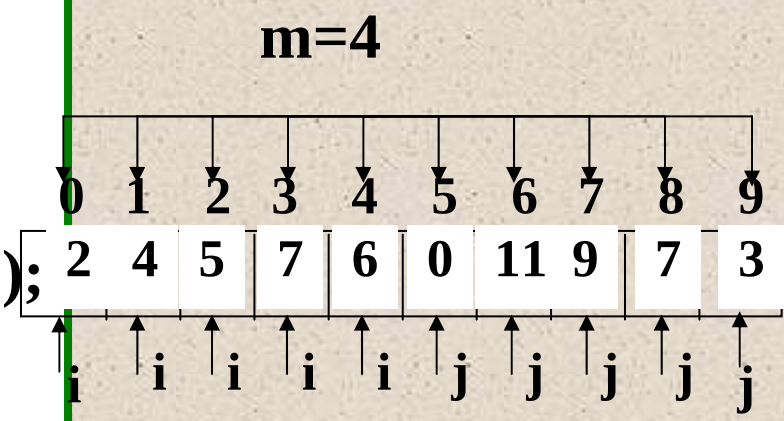
  invert(int s[],int i,int j)

  实现元素s[i]…s[j]的反置

//根据指针和数组关系， **int *a**为传数组

实参与形参均用数组名

```
#include <stdio.h>
void inv(int  x[], int n);
main()
{   int i,a[10]={3,7,9,11,0,6,7,5,4,2};
 inv(a,10);
 printf("The array has been inverted:\n");
    for(i=0;i<10;i++)
       printf("%d,",a[i]);
    printf("\n");
}
void inv(int  x[], int n)
{   int t,i,j,m=(n-1)/2;
    for(i=0;i<=m;i++)
    {    j=n-1-i;
            t=x[i];  x[i]=x[j];  x[j]=t;    }
}
```



实参用数组名
形参用指针变量

```
#include <stdio.h>
void inv(int  *x, int n);
main()
{   int i,a[10]={3,7,9,11,0,6,7,5,4,2};
    inv(a,10);
   printf("The array has been inverted:\n");
   for(i=0;i<10;i++)
      printf("%d,",a[i]);
   printf("\n");
}
void inv(int  *x, int n)
{   int t,i,j,m=(n-1)/2;
   for(i=0;i<=m;i++)
   {    j=n-1-i;
          t=x[i];  x[i]=x[j];  x[j]=t;    }
}
```

实参与形参都用指针变量

例10.7 将数组a中的n个整数按相反顺序存放

实参与形参都用指针变量

```
#include <stdio.h>
void inv(int  *x, int n);
main()
{   int i,a[10]={3,7,9,11,0,6,7,5,4,2};
    int  *p=a;  inv(p,10);
    printf("The array has been inverted:\n");
    for(i=0;i<10;i++)
       printf("%d,",a[i]);
    printf("\n");
}
void inv(int  *x, int n)
{   int t,i,j,m=(n-1)/2;
    for(i=0;i<=m;i++)
    {    j=n-1-i;
            t=x[i];  x[i]=x[j];  x[j]=t;    }
}
```

实参用指针变量
形参用数组名

```
void inv(int *x,int n)
  {int *p,temp,*i,*j,m=(n-1)/2;
   i=x;j=x+n-1;p=x+m;
   for(;i<=p;i++,j--)
   {temp=*i;*i=*j;*j=temp;}
   return;
}
```

例10.7 将数组a中的n个整数按相反顺序存放

实参用指针变量
形参用数组名

```
#include <stdio.h>
void inv(int  x[], int n);
main()
{   int i,a[10]={3,7,9,11,0,6,7,5,4,2};
    int  *p=a;  inv(p,10);
    printf("The array has been inverted:\n");
    for(i=0;i<10;i++)
       printf("%d,",a[i]);
    printf("\n");
}
void inv(int  x[], int n)
{   int t,i,j,m=(n-1)/2;
    for(i=0;i<=m;i++)
    {    j=n-1-i;
           t=x[i];  x[i]=x[j];  x[j]=t;    }
}
```

8.7 字符指针

- 字符数组和字符串常量

C++中的字面字符串有两种:

char buffer[]="hello";　　//用于字符数组初始化

cout<<"good"<<endl;　　//字符串常量,占有存储空间

指针与字符串

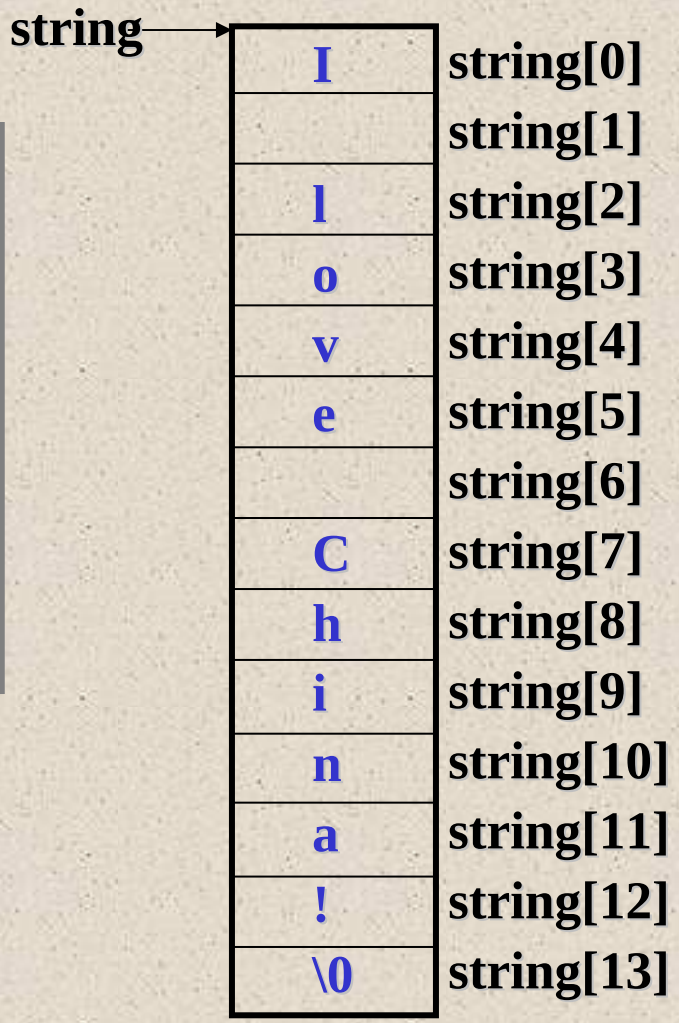
1、字符串表示形式

- 用字符数组实现

```
例:
void  main ( )
{
    char string[] = "I love China!";
    printf ("%s\n", string);
    printf ("%s\n", string + 7);
}
```

运行结果:

```
I love China!
China!
```



➢ 用字符指针实现

字符指针初始化：把字符串首地址赋给string

⇔ char *string;

string = “I love China!”;

```
例:
void main ( )
{
   char *string = “I love China!”;
   printf (“%s\n”, string);
   string += 7;
   while (*string)
   {
      putchar (string[0]);
      string++;
   }
}
```

整体引用

*string != 0

逐个字符引用

运行结果：

I love China!

China!

2、字符指针变量与字符数组

char *cp; 与 char str[20];

- str由若干元素组成，每个元素放一个字符；而cp中存放字符串首地址
- char str[20]; str="I love China!"; (×)

  char *cp; cp="I love China!"; (✓)
- str是地址常量；cp是地址变量
- cp接受键入字符串时,必须先开辟存储空间

例 char str[10];
scanf("%s",str); (✓)

而 char *cp;
scanf("%s", cp); (×)

改为：char *cp, str[10];
cp = str;
scanf ("%s",cp); (✓)

3、字符串与数组关系

- ✓ 字符串用一维字符数组存放
- ✓ 字符数组具有一维数组的所有特点
  - ● 数组名是指向数组首地址的地址常量
  - ● 数组元素的引用方法可用指针法和下标法
  - ● 数组名作函数参数是地址传递等
- ✓ 区别
  - ● 存储格式：字符串结束标志
  - ● 赋值方式与初始化
  - ● 输入输出方式：%s %c

```
scanf("%s",str);
printf("%s",str);
gets(str);
puts(str);
```

```
char  str[]={"Hello!"};          (√)
char  str[]="Hello!";            (√)
char  str[]={'H','e','l','l','o','!'};  (√)
char  *cp="Hello";               (√)
int   a[]={1,2,3,4,5};           (√)
int   *p={1,2,3,4,5};            (×)
```

```
char  str[10],*cp;
int   a[10], *p;
str="Hello";     (×)
cp="Hello!";     (√)
a={1,2,3,4,5};   (×)
p={1,2,3,4,5};   (×)
```

4、字符指针变量使用注意事项

当字符指针指向字符串时，除了可以被赋值之外，与包含字符串的字符数组没有什么区别。

```
char str[10], *pstr;
pstr = "12345";     //pstr指向"12345"
strcpy (str, pstr);  //将pstr所指向的字符串复制到数组str中
pstr = str;
printf ("The Length of str is: %d\n", strlen(pstr));  //输出字符串的长度5
```

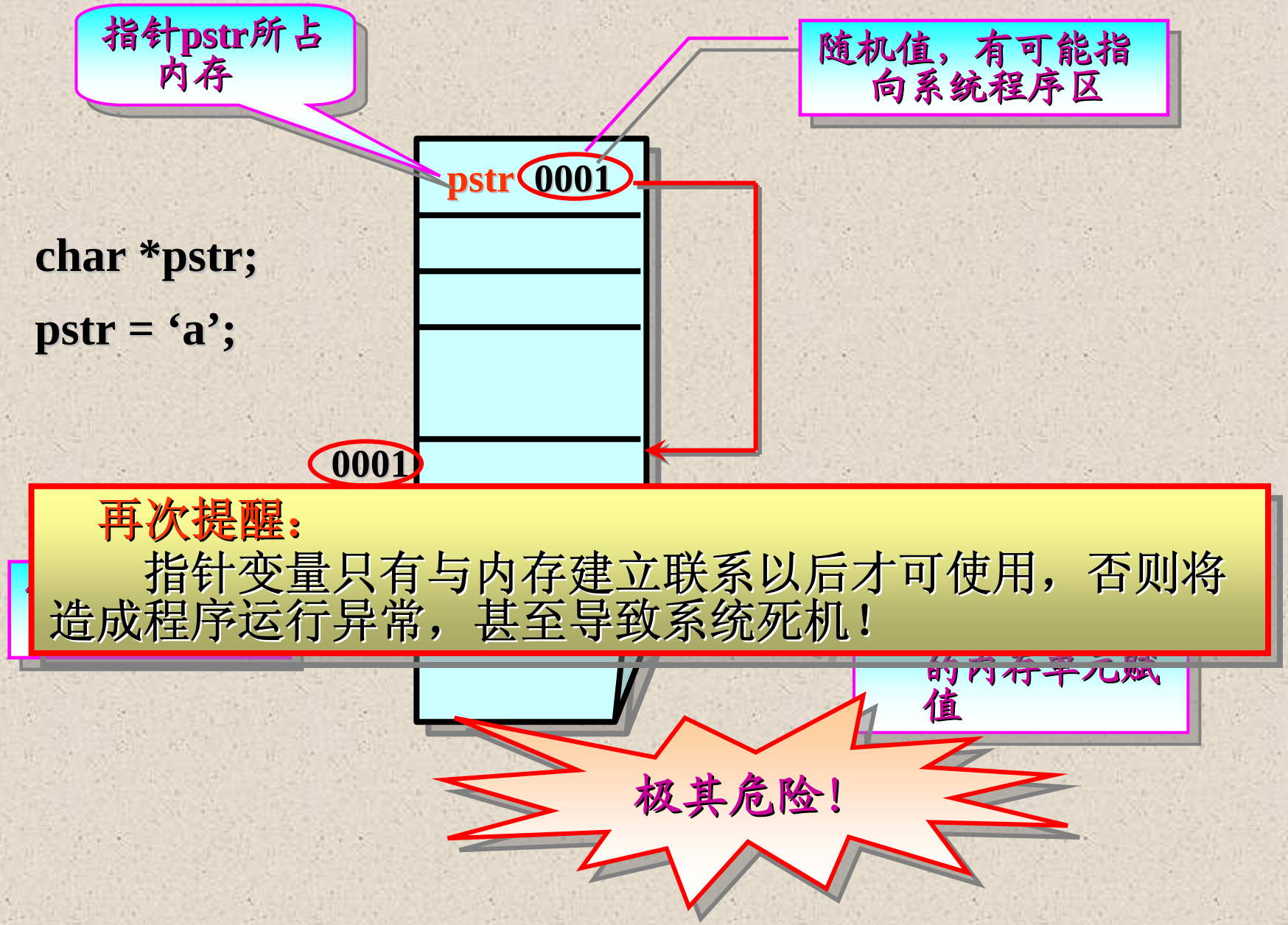
注意“野指针”操作：

如果一个指针[illegible]被称为“野指针”操作[illegible]会出错，但很容易[illegible]崩溃。

为什么“野指针”操作会给程序运行带来极大的不确定性，甚至造成系统崩溃呢？

```
char *pstr;
char str[8];
scanf ("%s",pstr);          //野指针操作，pstr没有指向有效内存
strcpy (pstr, "hello");     //野指针操作
pstr = str;    //pstr指向数组str所对应内存单元的首地址
strcpy (pstr, "0123456789"); //不是野指针，但会造成数组越界
```

为什么“野指针”赋值会给程序运行带来极大的危险？



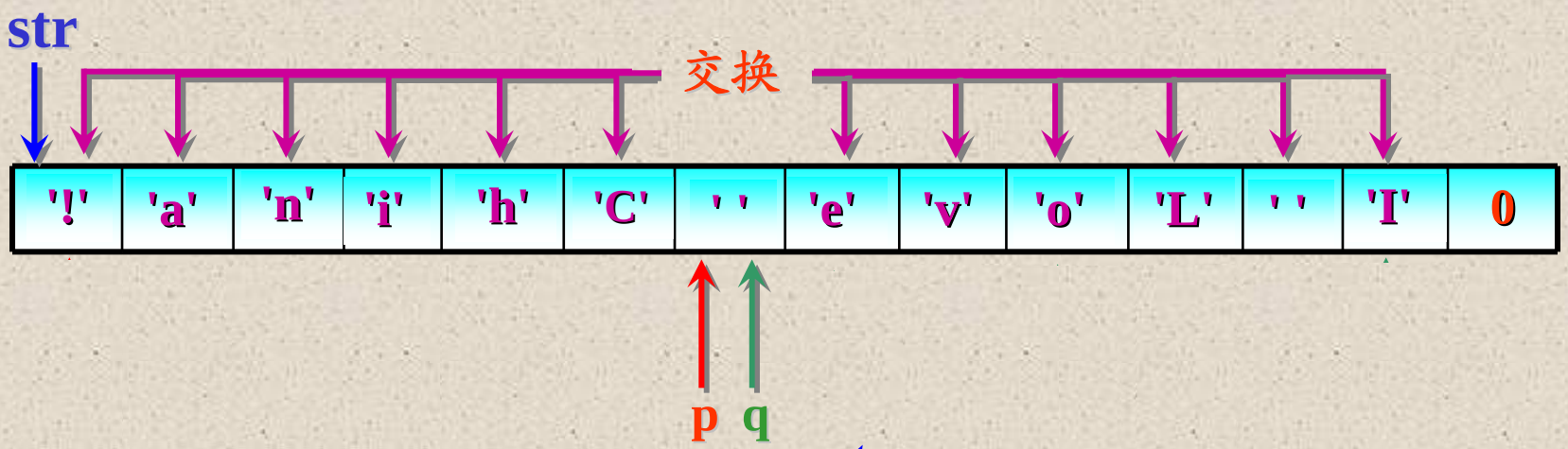
【例】 利用字符指针实现字符串的倒序排列

```
#include <stdio.h>
#include <string.h>
void main ( )
{
  char str[200], ch;
  char *p, *q;
  gets (str);                    //读取一个字符串
  p = str;                       //p指向字符串的首地址
  q = p + strlen(p) - 1;         //q指向字符串的末地址
  while (p < q)
  {                              //交换p和q各自指向的字符
    ch = *p;                     //将p所指向的字符保存在ch中
    *p++ = *q;                   //先将q指向的字符赋给p指向的字符单元，p再增1
    *q-- = ch;                   //先将ch的值赋给q指向的字符单元，q再减1
  }
  printf ("%s\n", str);
}
```

运行结果：

```
I love China! ↙
!anihC evol I
```

程序执行过程演示：



p = str;

End

q = p + strlen(p) - 1

字符串比较

```
void main()
{     char buffer1[10]="hello";
      char buffer2[10]="hello";

      if(strcmp(buffer1,buffer2)==0)      //比较字符串的内容
            cout<<"They are same."<<endl;
      else
            cout<<"They are different."<<endl;

      if(buffer1==buffer2)                //比较字符串的首地址
            cout<<"They have the same address."<<endl;
      else
            cout<<"They have the different address."<<endl;
}
```

- 可以将字符串复制到字符数组，但不可以复制到字符指针:

```
char buffer[10];
strcpy(buffer,“hello”);          //ok

char *pstr;
strcpy(pstr,"hello");            //error
```

练习

```
point(char  *p)
{  int i;
   while(*p!='\0')
   {  *p+=3;
      p++;
   }
}
main()
{   char  b[100]="good",*p=b;
    point(p);
    printf("%s\n",p);
}
```

```
while(*p!=0)
或
while(*p)
```

运行结果：
jrrg

练习 输出结果

```
#include <stdio.h>
#include <string.h>

void func (char *s);

void main ( )
{
  char str[80];

  printf ("Input string: ");
  gets (str);
  func (str);
  printf ("After delete digital
    char: %s\n", str);
}
```

```
void func (char *s)
{
  char *pstr;

  for (pstr = s; *pstr != '\0';
    pstr++)
  {
    while (*pstr >= '0' && *pstr
    <= '9')
        strcpy (pstr, pstr+1);
  }
}
```

输入:21TYPE5TEST78

练习

- 编程判断输入的一串字符是否为“回文”
- **Eg: level　acca　　(5分钟完成)**

练习

- 编程判断输入的一串字符是否为“回文” exc11_5.cpp

```
#include <stdio.h>
#include <string.h>
void main ( )
{  char str[80], *pStart, *pEnd;
  int len;
  printf ("Input String: ");
  gets (str);
  len = strlen (str);
  pStart = str;
  pEnd = str + len - 1;
  while (*pStart == *pEnd &&
          pStart <= pEnd)
  {       pStart++;
          pEnd--;
  }
  if (pStart < pEnd)
          printf ("No!\n");
  else
          printf ("Yes!\n");
}
```

多级指针的定义和应用

1. 二级指针变量的定义

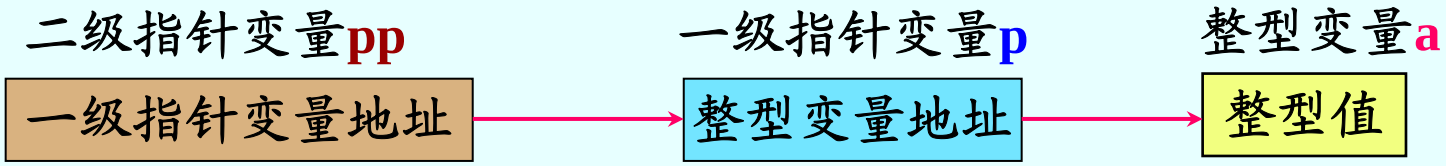
二级指针变量定义的一般形式：

数据类型标识符 ****指针变量名**;

“数据类型标识符”是最终目标变量的类型。

例如：

int a, *p, **pp;

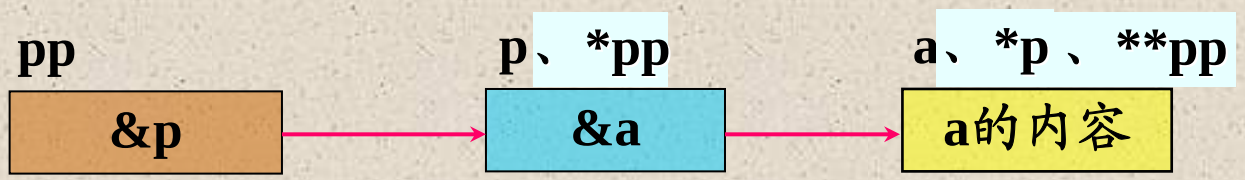


多级指针的定义和应用（续）

2. 二级指针变量初始化

例如：

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



多级指针定义形式

定义形式：[存储类型] 数据类型 **指针名;

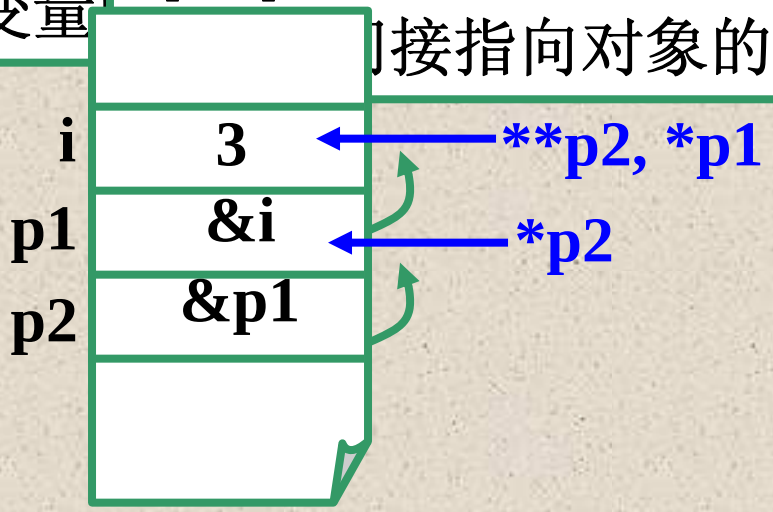
如 char **p;

*p是p间接指向对象的地址

最终目标变量

接指向对象的值

```
例 int  i=3;
    int  *p1;
    int  **p2;
    p1=&i;
    p2=&p1;
    **p=5;
```



例 int i, **p;

p=&i; (×)//p是二级指针，不能用变量地址为其赋值

多级指针

例 三级指针 int ***p;

四级指针 char ****p;

本章小结

- 指针的概念
- 指针运算：**p+n、p-n、p1-p2、p1>p2……**
- 指向数组的指针
- 动态分配内存：**new 、delete**
- 指针与函数
  - 指针做函数参数
  - 返回指针的函数
- 字符指针

练习

- 编程判断输入的一串字符是否为”回文”
- **Eg: level　acca　　(5分钟完成)**

希望大家能学出好成绩，我们一起努力！

谢谢大家！