Chapter 9 Pointer(指针)

pointer：The elite of C,import concept, import feature.

Using pointer：

1.can make program more compact,

2.be care of using pointer

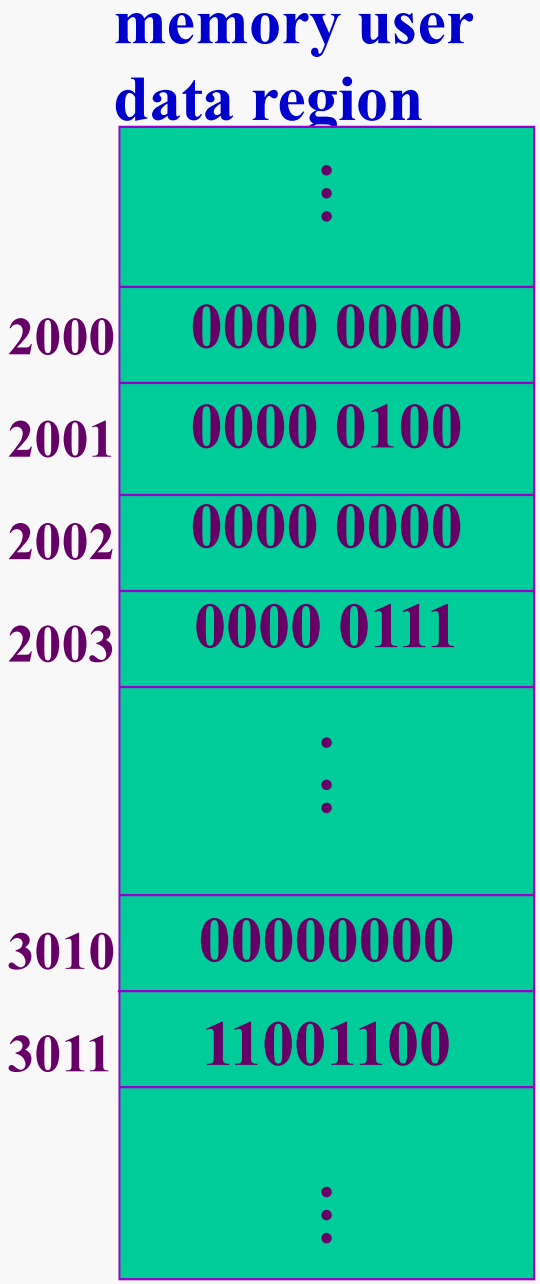
9.1 The concept of the pointer and pointer variable

Pointer is address

address

1. the address of the memory storage unit ——the serial number of the storage unit

cf.data storing in the memory storage unit

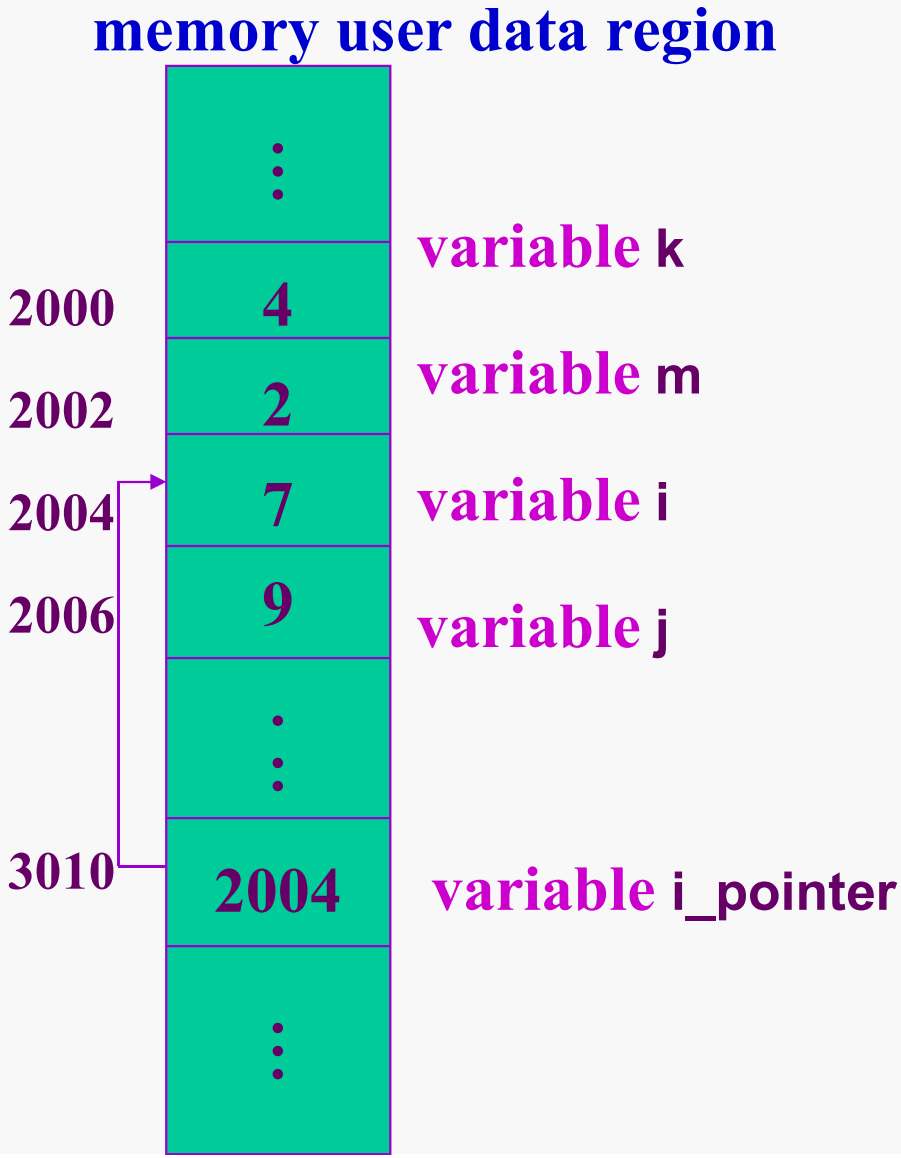


2. Variable address and value

(1) variable address——
The address of a variable is itself data that can be manipulated and stored in memory

(2) value——can be stored in memory

(3) lvalue



How does the compiler process variable definition ?

```
main()
    { int num;
      scanf("%d",&num);
      printf("num=%d\n", num);
    }
```

C compiler does operate as follows when it compile variable definition statement ” int num”:

(1) distribute variable memory space

(2) login variable num to "symbol table"

each variable in symbol table contains two attribute：

1) variable name（id）

2) its address （addr）

3. The access of variable value

(1)Direct access

```
scanf("%d",&m);
printf("%d", m);
m=m+i;
```

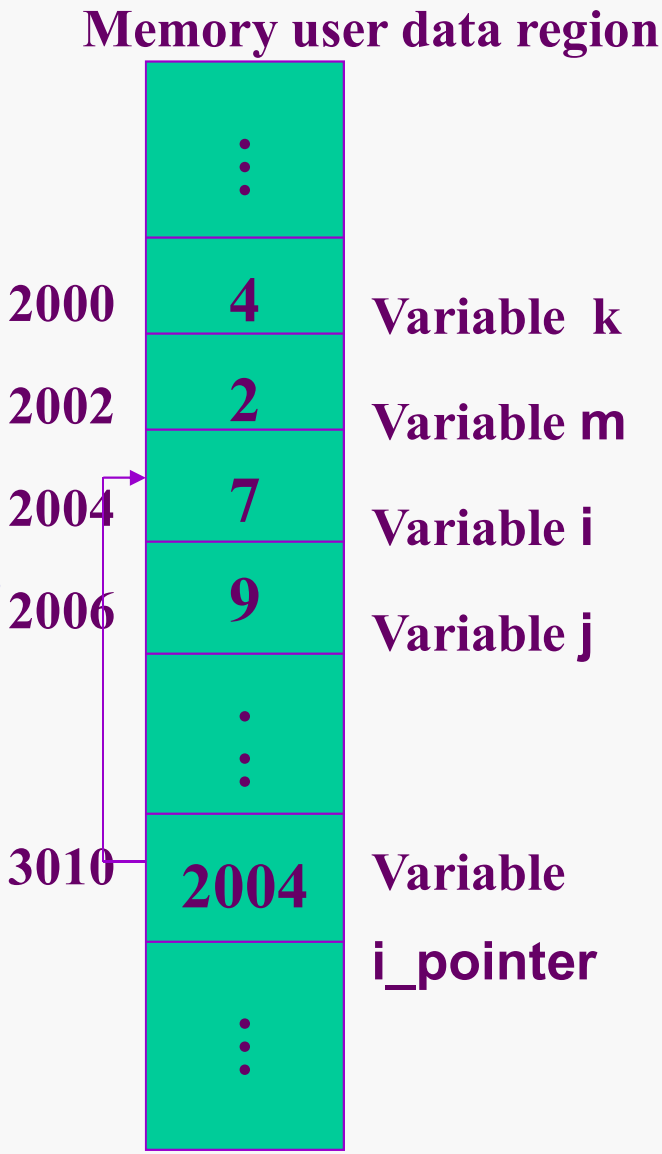
(2) indirect access

visit variable value through another variable (a pointer variable)

Example:

define a pointer" i_pointer"

```
i_pointer=&i ;
scanf("%d",i_pointer);
```



9．2 Pointer variable points to a variable

The definition of pointer variable

general format：

base-type *pointer-variable ;

where： base-type is the type of the value to which the pointer points

pointer-variable is the variable being declared

for examples, int *p1; char *p2;

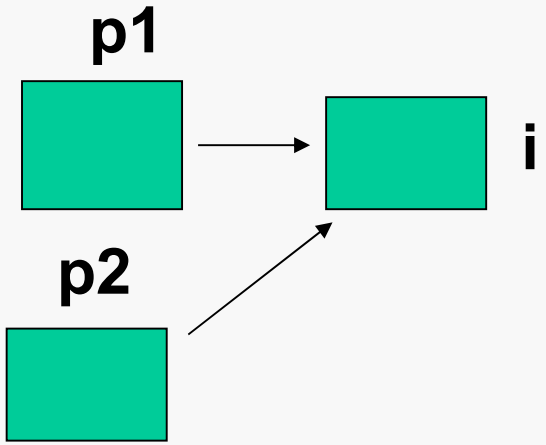
using a pointer variable

1. assignment of a pointer variable value (to assign an address to a pointer)

for example, int *p1,*p2, i;

p1=&i;

p2=p1;



2. to assign a value to a variable with pointer variable

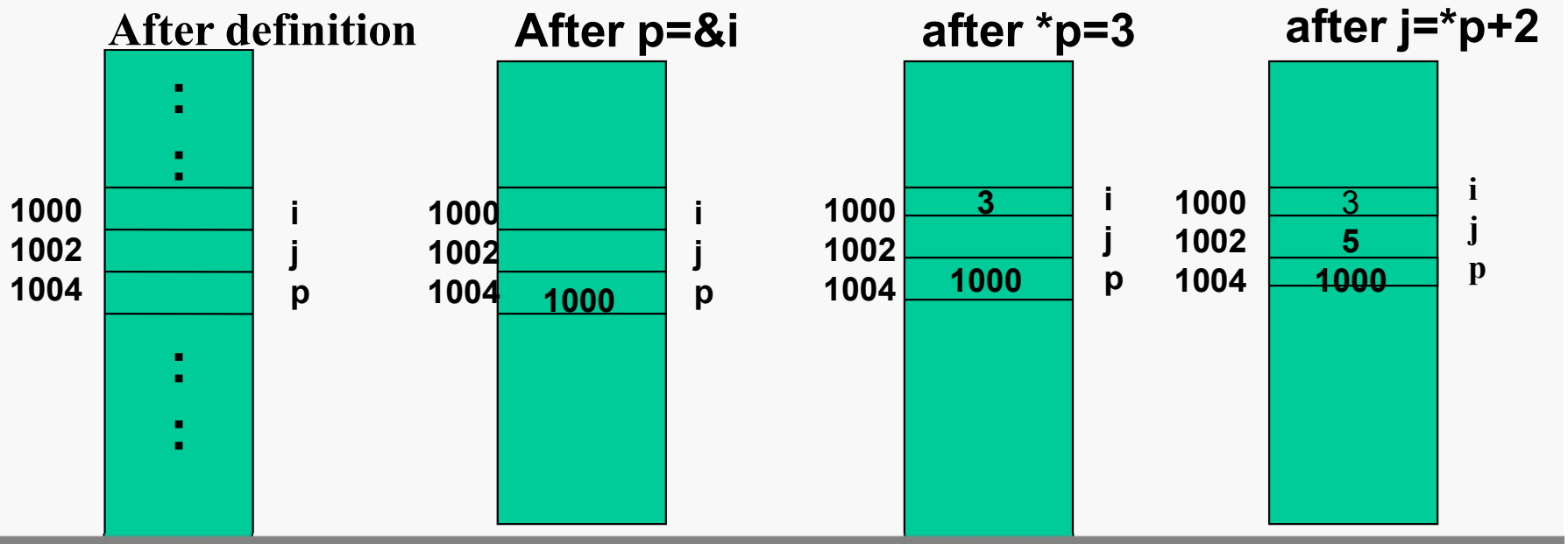
"*": pointer operator、indirection operator is an unary operator(value-pointed-to)

for example ,int i , j , *p;

p=&i;

***p=3;** **namely i=3;**

j=*p+2;



Pointer variable must be assigned a value before it is used ,or it becomes a hanging pointer!

*operator & and * :*

（1）associativity :from right to left

if：int a, *p1;

p1=&a;

then： &*p1 =&(*p1)=&(a)=&a=p1

&a =(&a)=a

（2）priority

* prior to &

3.”--”and “++”operator mostly apply to arrays’ pointer

if ： float x,*p1; p1=&x;

if p=1000, then p++ is 1004

example： int a,b,*p;

p=&a; a=3; b=5;

(*p)++; the equivalence is a++ a==4

*p++; the equivalence is *(p++)

if p=1000, then p++ is 1002

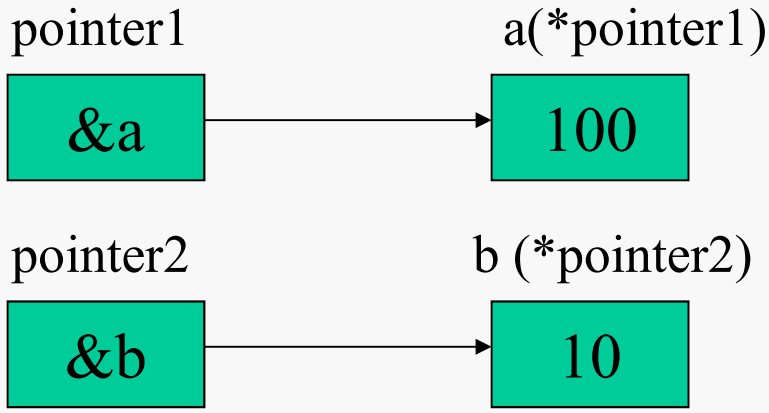
[9.1]The indirect access of variables

```
main()
 {  int a，  b;
    int  *pointer_1,  *pointer_2;
    a=100；b=10；
    pointer_1=&a；  /*to assign  a's address to pointer_1*/
    pointer_2=&b；/* to assign  b's address to pointer_2*/
    printf  （”％d，％d\n”, a，b);
    printf（”％d，％d\n”, *pointer_1，*pointer_2）；
 }
```

The results：

100, 10

100, 10



[9.2]Input two integers a and b ,output them ordered by value .

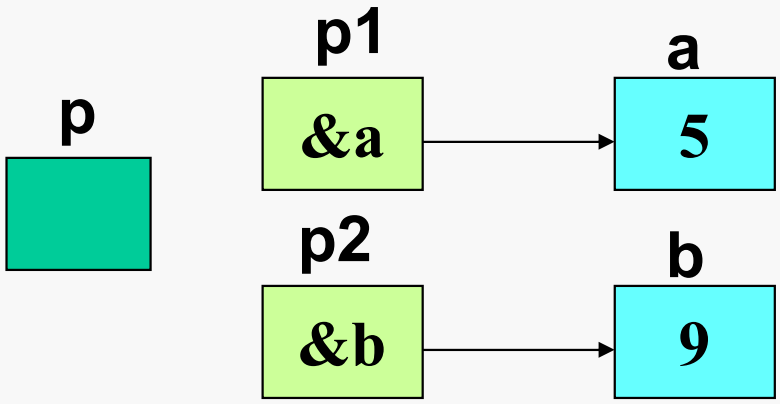
```
main()
        { int *p1，*p2，*p, a, b;
          scanf （"%d%d"，&a，&b）;
          p1=&a;  p2=&b;
          if (a<b)
              {p=p1；pl=p2；p2=p;}
          printf （"\na=%d,b=%d\n”，a，b）;
          printf（"max=%d,min=%d\n"，*pl，*p2）;
        }
```

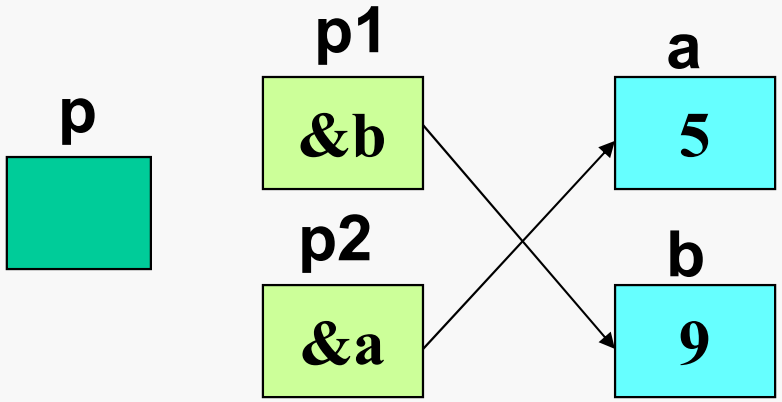
if(*p1<*p2)

results： 5 9↓

a=5，b=9

max=9，min=5


p
p1
&a
a
5
p2
&b
b
9


p
p1
&b
a
5
p2
&a
b
9

Modify program as follows:

```
main()
 {  int *p1，*p2，p, a, b;
     scanf （"%d%d"，&a，&b）;
      p1=&a;  p2=&b;
      if (*p1＜*p2)
         {p=*p1；*pl=*p2；*p2=p;}
      printf （"\na=%d,b=%d\n”，a，b）;
      printf（"max=%d,min=%d\n"，*pl，*p2）;
}
```

question：**1.analyses function result**

5 9↓

a＝?，b＝?

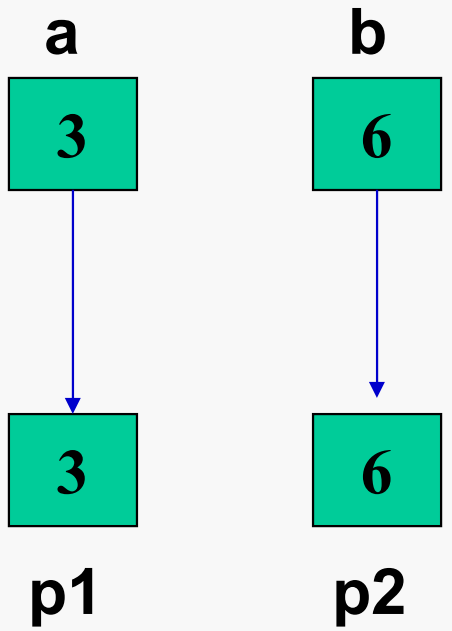
max＝?，min＝?

**2.how about add three *before *p*?

4.pointer variable act as function parameter

question：can the swap function able to realize two variable exchange?

```
void  swap(int p1,int  p2)
{  int t;
   t=p1; p1=p2; p2=t;
}
main()
{   int a,b;
    printf("a,b=");
    scanf("%d%d",&a,&b);
    swap(a,b);
    printf("a=%d,b=%d\n",a,b);}
```

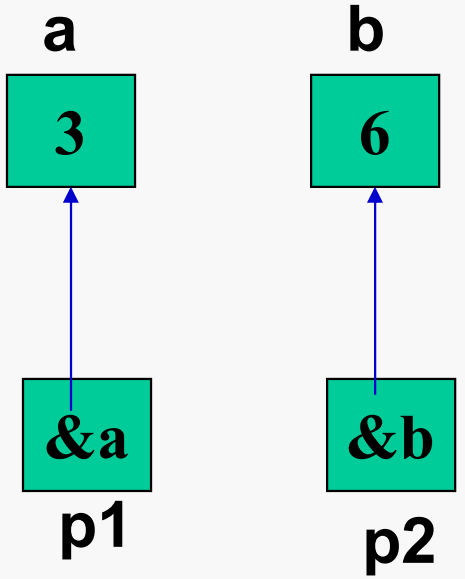


[9.3] *pointer variable act as function parameter*

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

main()
{   int a,b;
    printf("a,b=");
    scanf("%d%d",&a,&b);
    swap(&a,&b);
    printf("a=%d,b=%d\n",a,b);
}
```

Transfer value(address)



Summary:

We must use pointer variable as function parameter to return the alternative value to calling function from called function

Mechanism:

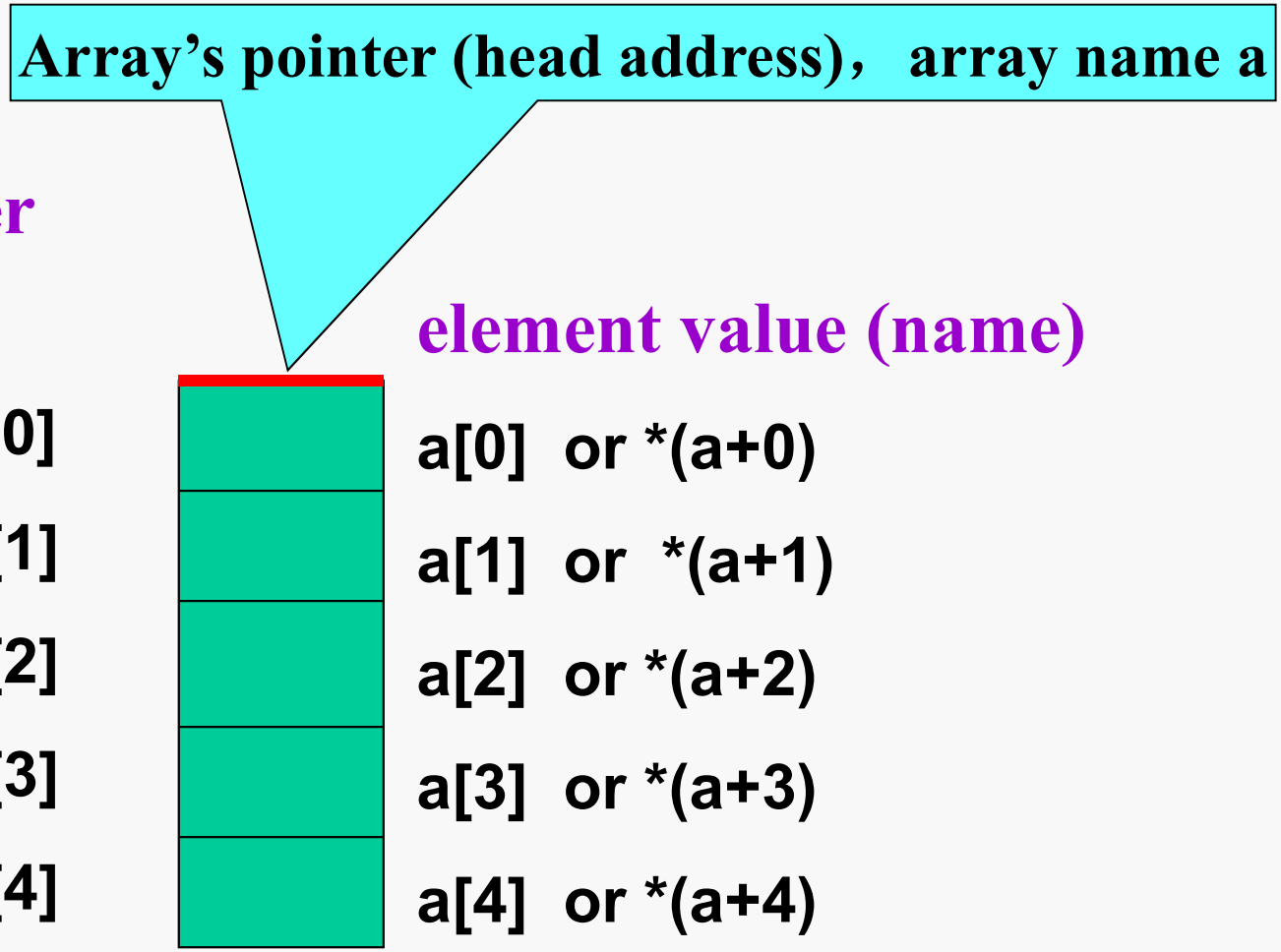
When system execute the called function , if the formal parameter pointer variable changes, the value is saved via actual parameter at the end of the function call.

9.3 Pointer points to an array

1. summarization

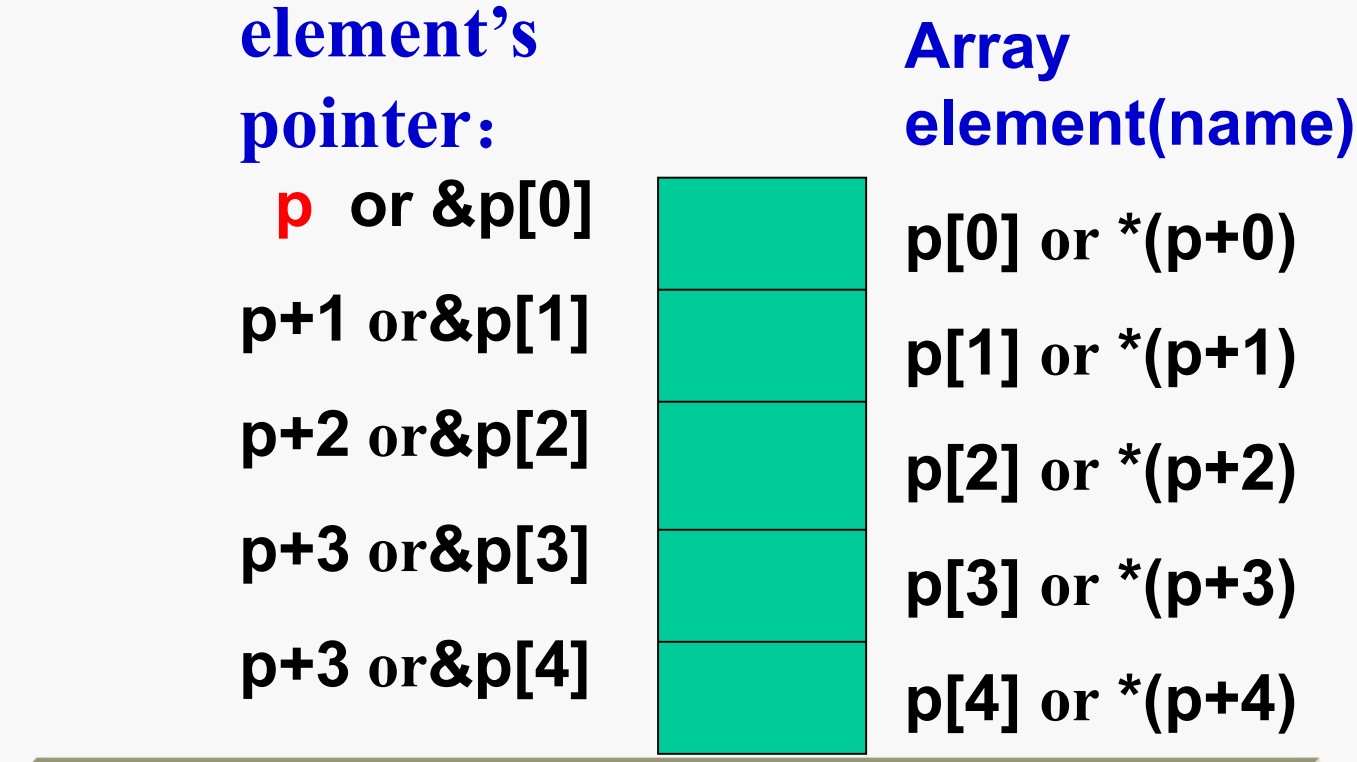
•conception

int a[5];



- **Declare a pointer of one-dimensional array**

For example, int a[5], *p=a (or *p=&a[0]);



instruction: int a[5], *p=a (or&a[0]);
namely: int a[5], *p;
p＝a;

- **Quoting one-dimensional array element**

	element value	element address	
Subscript method:	a[i]	&a[i]	visual

Pointer method :	*(a+i)	a+i
	*(p+i)	p+i

object program occupys memory fewer and run fast.

- **The difference between p and a:**

  a: address constant

  p: address variable

[9.5] The quotation of array 's elements

First:subscript method

```
main()
{    int a[10],  i;
     printf("Input 10 numbers: ");
     for(i=0; i<10; i++)
         scanf("%d", &a[i]);
     printf("a[10]: ");
     for(i=0; i<10; i++)
         printf("%d  ", a[i]);
     printf("\n");
}
```

Second： array name

```
main()
{   int a[10],  i;
    printf("Input 10 ...: ");
    for(i=0; i<10; i++)
        scanf("%d", a+i);
    printf("a[10]: ");
    for(i=0; i<10; i++)
       printf("%d", *(a+i));
    printf("\n");
}
```

results：

```
Input 10 numbers: 0 1 2 3 4 5 6 7 8 9↵
a[10]:  0  1  2  3  4  5  6  7  8  9
```

third： pointer variable

```
main()
{  int a[10], i, *p=a;
   printf("Input 10 ... : ");
   for(i=0; i<10; i++)
       scanf("%d", p+i);
   printf("a[10]: ");
   for(i=0; i<10; i++)
      printf("%d  ", *(p+i));
   printf("\n");
}
```

four： pointer variable ++

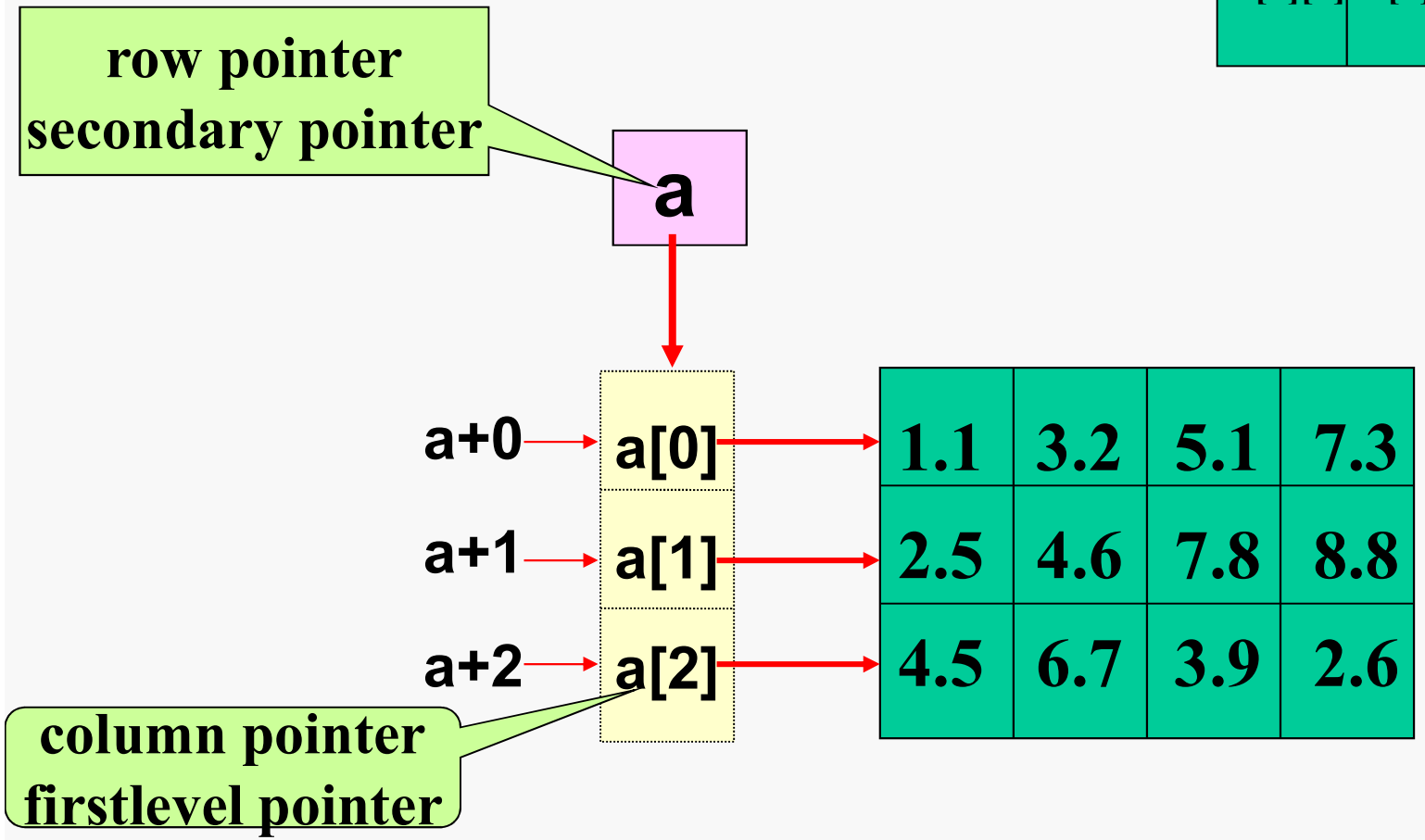
```
main()
{  int a[10],  *p;
   printf("Input 10 ... : ");
   for(p=a; p<a+10; p++)
       scanf("%d", p);
   printf("a[10]: ");
   for(p=a; p<a+10; p++)
      printf("%d  ", *p);
   printf("\n");
}
```

pay attention to the difference between p+1 and p++!

3. Two-dimensional array's pointer

Two –dimensional array's pointer :
example: int a[3][4];

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]



discription：two_dimentional array has some features :

- array name “a”stands for the starting address and a row pointer that controls a line.

a+i： row pointer， points to line i

*(a+i)or a[i]: col pointer， pointer to line i column 0

- the format of how to access a[i][j] with pointer.

a[i][j]

*(a[i]+j)

((a+i)＋j)

(*(a+i))[j]

analyses：

- **a[i]+j：　column pointer points to array element a[i][j].**

- ***(a[i]+j)：　the value of array element a[i][j] .**

- **If　int a[3][4], *p=a[0]；**

  so p+1point to ？

  how to use p as pointer to access a[i][j]?

4.Row pointer variable

—a pointer variable points to an one-dimensional array consists of N elements

definition form

data type (* row pointer variable) [n];

notice：() indispensability，or it becomes pointer array.

assign a value

row pointer variable= two dimensional array name| row pointer variable;

[9.6] Using row pointer and column pointer to output any elements in the two _dimentional array.

- **using row pointer variable:**

```
 main()
{   int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    int (*p)[4], row, col;
    p=a;     printf("Input row = "); scanf("%d", &row);
    printf("Input col = "); scanf("%d", &col);
    printf("a[%d][%d] = %d\n", row, col, *(*(p+row)+col));}
```

results：

Input row = 1↵

Input col = 2↵

array[1][2] = 7

reflect：we can also use array name A as pointer， how to modify the program？

•using column pointer

```
main()
  { int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
    int *p, row, col;
    p=a[0];
    printf("Input row = "); scanf("%d",&row);
    printf("Input col = "); scanf("%d",&col);
    printf("a[%d][%d]=%d\n",
                row,col, *(p+(row*4+col)));
  }
```

5. The realization of the dynamic array

static array：

The size of the array during the program executing can not be changed.

disadvantage: waste the memory space if we can not to estimate the quantity of sata

dynamic array:

Specify the size of the array according to actual requirement during the program executing.

In c language,we can realize the dynamic array by using array's pointer act as array name and library functions that apply or release memory.

dynamic array's essence： one pointer points to an array

[9.7] The realization of the dynamic array

```
#include  “alloc.h”
#include  “stdlib.h”
main()
{ int  *array=NULL, num, i;
   printf(“Input the  number  of   element: ”);
   scanf(“%d”, &num);
   /*block memory used as application dynamic array */
   array=(int  *)malloc( sizeof(int) * num );
   if ( array==NULL )
    /* applying memory space is failed：prompt，exit.*/
    { printf(“out  of  memory, press  any  key  to  quit……”);
      exit(0);
   /*exit()：terminate running，return Operating System*/
     }
```

```
printf("Input  %d  elements: ", num); /*prompt to input the number of integer*/
 for (i=0; i<num; i++) scanf("%d", &array[i]);
  /*prompt to output the number of integer*/
printf("%d  elements  are: ", num);
for (i=0; i<num; i++) printf("%d,", array[i]);
    printf("\b ");    /* delete the last one data rear separator",  " */
    free(array);
/*release the memory block from the application of  malloc() function*/
}
```

results：

Input the number of element: 3↵

Input 3 elements: 1 2 3↵

3 elements are: 1,2,3

program description：

array=(int *)malloc(sizeof(int) * num); statement

——malloc() function and sizeof operator

•library function malloc()

usage：void *malloc(unsigned size)

function：distribute 1 series space in the dynamic storage area of the memory which length id size

return value：if application is success ,return the starting address ,if not return NULL。

·function prototype alloc.h，stdlib.h。

Malloc () returns a typeless pointer and it can points to any type data in order to void mistakes we must transfer the return value into the data type that points to .

•**operator sizeof**

form：sizeof(variable name ／type name)

function：calculate the space variable／type occupy.e.g，in IBM-PC sizeof(int)=2

•**free(array);statement——library function free()**

usage：void free(void *ptr)

function：release the memory block ptr points to.

return value：null

In principle,we use malloc () function to apply for MB and use free() function to release the MB after operation.If we release the MB out of time ,it may exhaust systemic memory resource soon , thereby the program can not run .

reflect：

（1）use sizeof(int) to calculate the memory byte number that int number ocuppy , why not to use constant 2?

（2） scanf(“%d”, &array[i]); printf(“%d,”, array[i]);

Pointer points to array acts as array name it must be used according to the syntax rule of quoting array elements.

（3） printf(“\b ”);statement