

Ch-06

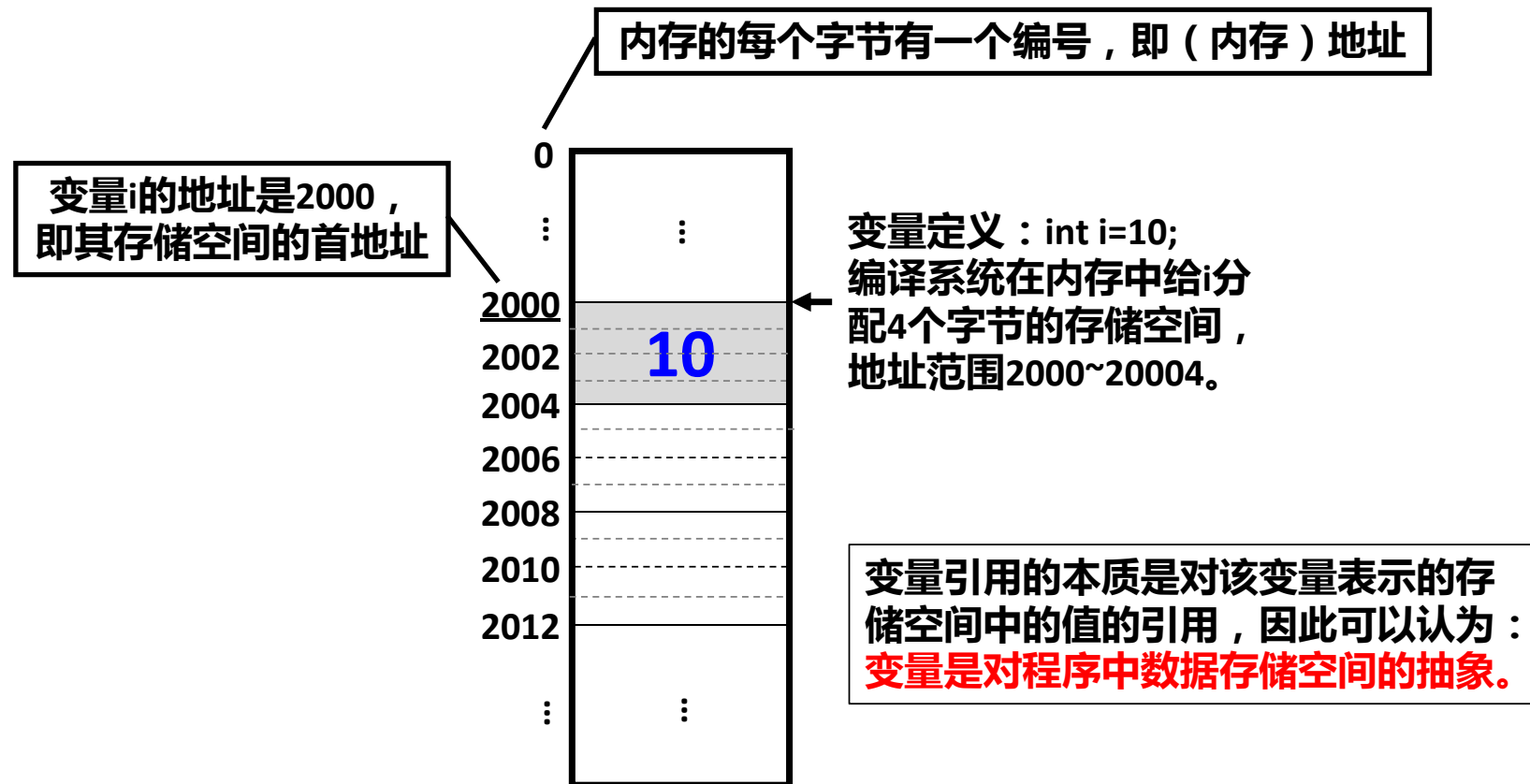
指 针

主要内容：

- 指针的概念**
- 指针变量的定义和引用**
- 指针与数组**
- 指针与字符串**
- 指针与函数**
- 指针数组和指向指针的指针**
- const指针和void类型指针**
- 引用**

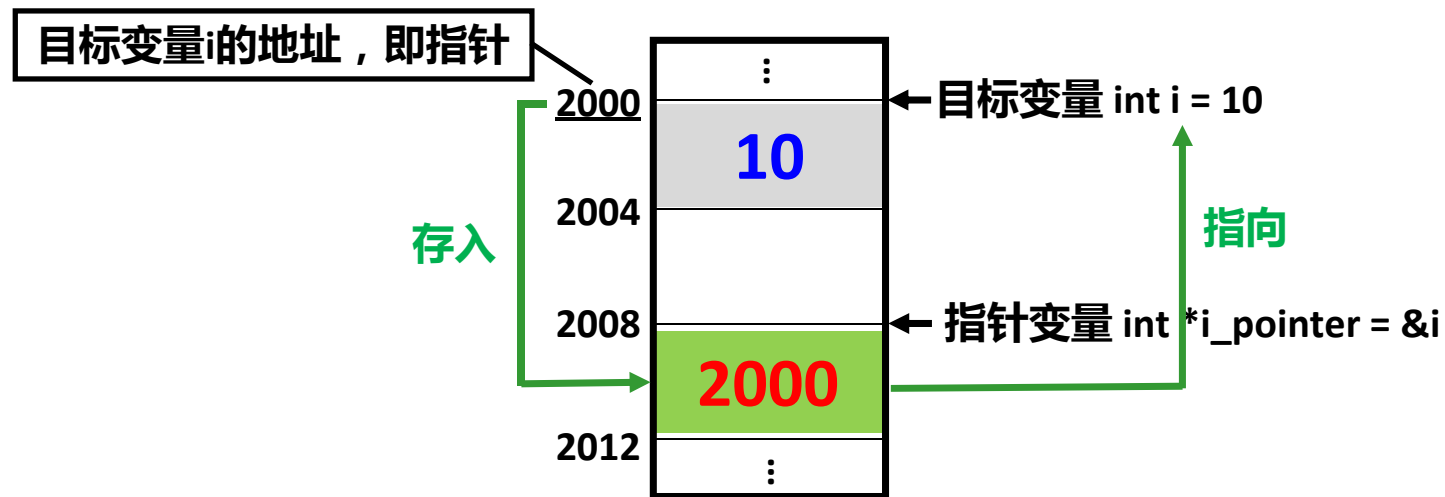
6.1 指针的概念

□ 变量与地址



□ 指针与指针变量

- 指针是（目标）变量的地址；
- 指针变量是专门存放指针（目标变量地址）的变量。



6.2 指针变量的定义和引用

□ 指针变量的定义形式

基类型 *指针变量名

- **说明：**1、基类型是指针指向的目标变量的数据类型；
2、*表示定义的变量是指针类型，即指针变量。
- **举例：**
`int *p1, *p2;`
`char *name;`
- **注意：**
 - 1、`int *p1, *p2;` 与 `int *p1, p2 ;`
 - 2、指针变量名是`p1, p2` ,不是`*p1, *p2` ;
 - 3、指针变量只能指向定义时所规定类型（基类型）的变量。

□ 指针变量的引用形式

引用形式	含义
指针变量名	引用该指针变量自身
*指针变量名	引用指针指向的目标变量

- 举例：已知 `int a;`

`int *p;`

则 `p = &a;` //指针变量p自身的引用，&a表示取变量a的地址

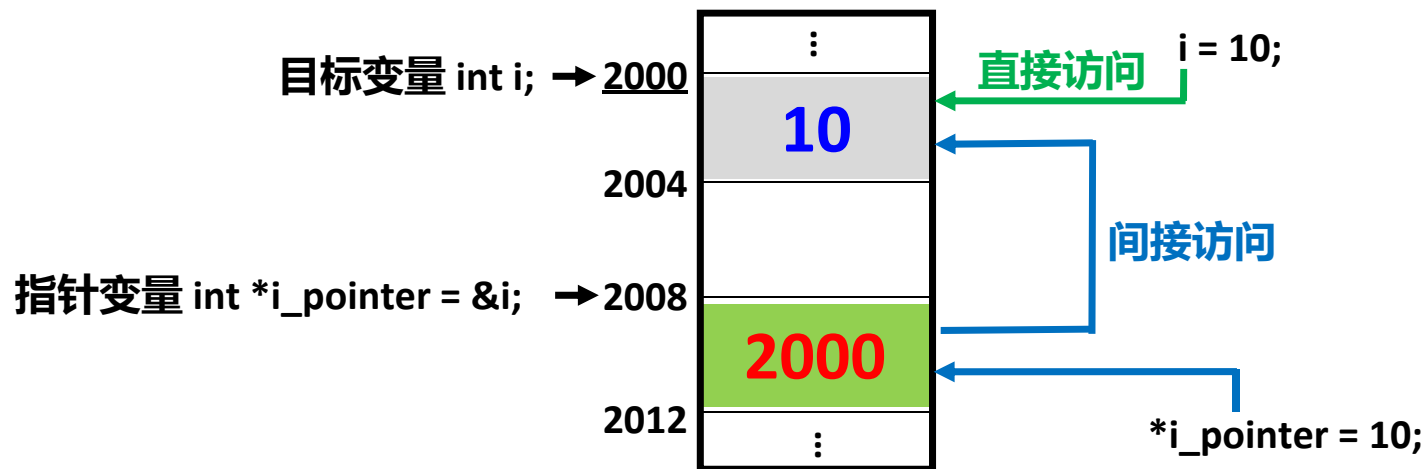
`*p = 5;` //目标变量a的引用，即*p等价于a

- **&运算符与*运算符**



运算符	&	*
名称	取地址运算符	间接访问运算符
含义	取变量的地址	取指针指向的目标变量的值
优先级	3	
结合性	从右向左	
两者关系	互为逆运算	

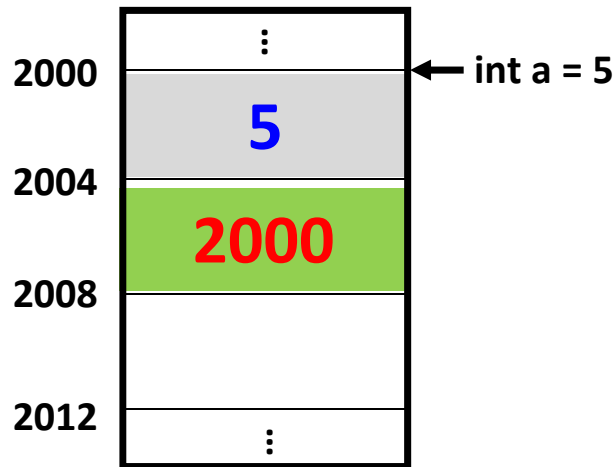
- 直接访问与间接访问

- 直接访问：根据变量地址直接存取变量值；
- 间接访问：首先通过指针变量的值获得目标变量的地址，然后存取目标变量的值。



练习1：已知int a=5, *p=&a; 且存储空间分配如图所示，求下列表达式的值和p的值。

- 1、 *p++  $\leftrightarrow *(p++)$ ，表达式的值是5，p的值是2004
- 2、 (*p)++  表达式的值是6，p的值是2000



练习2：若有 `int i = 10; int *i_pointer = &i;`

1、请分析下列表达式的含义？

`i_pointer`、`*i_pointer` 和 `&i_pointer`

`i_pointer`----**指针变量，值是地址**

`*i_pointer`----**指针指向的目标变量，值是普通数据**

`&i_pointer`---**指针变量的地址**



2、判断下列哪些表达式的值相等？

`i`、`i_pointer`、`&i`、`*i_pointer`、`*(&i)`、`&(*i_pointer)`

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

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



□ 指针变量的赋值

- 取已定义变量的地址进行赋值

例如：`int a, *p; p = &a;` 或 `int a, *p = &a;`

- 用已初始化的指针变量进行赋值

例如：`int a, *p, *k; p = &a; k = p;`

注意：

- 1、参与赋值运算的操作数的类型（即指针变量的基类型或普通变量的数据类型）必须相同；
- 2、指针变量中只能存放地址（指针），因此不要将整数（或任何其他非地址类型的数据）赋给指针变量。
- 3、指针变量定义后，必须先初始化再引用。

练习：请指出错误的地方，并改正。

1、`int *p = &a; int a = 6;`

2、`int *p; *p = 3;`

3、`int *p = 3;`

4、`int *p; float a; p = &a;`

5、`int a, b; int* p1, p2; p1 = &a; p2 = &b;`



1、`int a = 6; int *p = &a; //必须用已定义变量的地址对指针变量赋值`

2、`int a, *p=&a; *p = 3; //指针变量必须先初始化再引用`

3、`int a, *p=&a; *p = 3; //指针变量不能用非地址类型的数据初始化`

4、`float *p; float a; p = &a; 或 int *p; int a; p = &a; //类型要求相同`

5、`int a, b; int *p1, p2; p1 = &a; p2 = b; //p2是普通int型变量`

或 `int a, b; int *p1, *p2; p1 = &a; p2 = &b; //p2是指针变量`

6.3 指针与数组

□ 概念

- 数组的指针：数组的起始地址。
- 数组元素的指针：数组元素的地址。

□ 数组元素的引用

- 下标法： $a[i]$
- 指针法：指向数组的指针+偏移量。

6.3.1 指向一维数组元素的指针变量

定义数组 `int a[10];` 则

```
int *p;
```

```
p=&a[0]; /*等价于p=a; */
```

或 `int *p=&a[0];`

或 `int *p=a;`

说明：

- 1、`a`是数组名，表示数组首地址的地址常量；
- 2、不能用整数给`p`赋值，也不能用`p`的值给整型变量赋值。

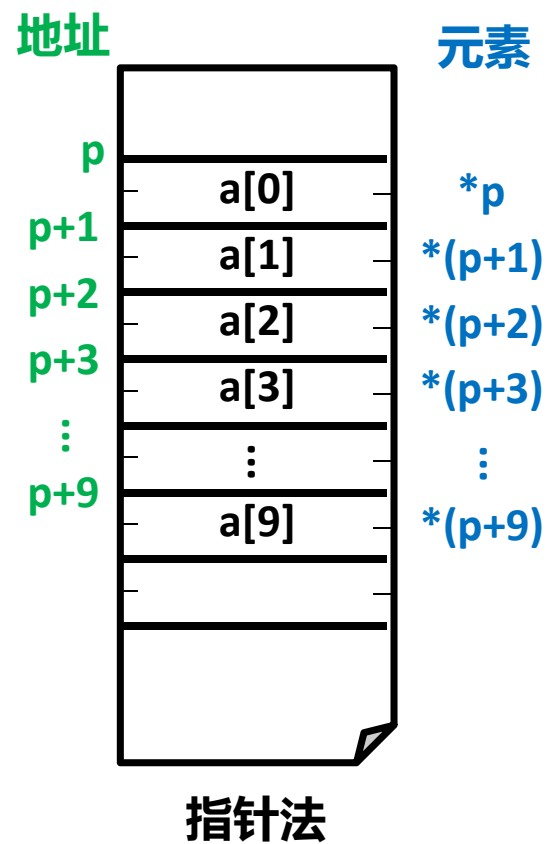
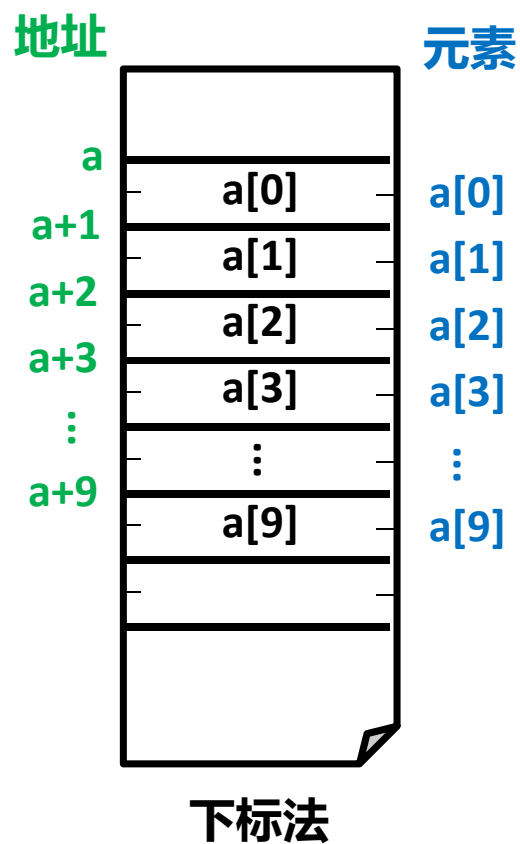
如 `int i, *p;`

`p=1000;` (×)

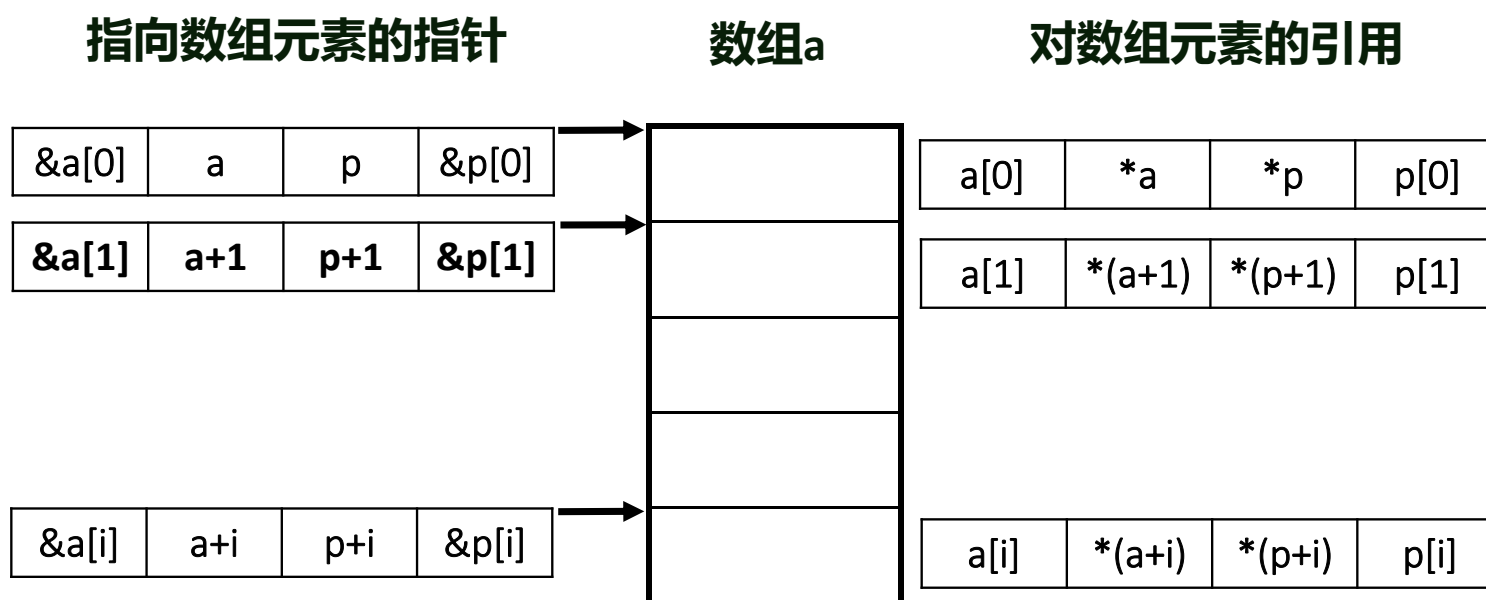
`i=p;` (×)

6.3.2 通过指针引用一维数组的元素

若 `int a[10], *p=a;` 则数组元素有两种引用方法：



因此，通过指针引用一维数组元素时存在以下等价形式：



□ 指针的算术运算

- $(p \pm i)$ 指向 p 前面或后面的第 i 个元素，即第 i 个元素的地址；
 - 若 p_1 与 p_2 指向同一数组，则 $(p_1 - p_2)$ =两指针间的元素个数；
 - $p_1 + p_2$ 无意义。
-

练习：

- 1、`int a[10], *p=a;` 则 $p+1$ 指向哪个元素；
- 2、`int a[10]; int *p=&a[2];` 则 $*p++=1$ 是给哪个元素赋值, p 指向哪个元素;
- 3、`int a[10]; int *p1=&a[2]; int *p2=&a[5];` 则 $p_2 - p_1$ 的值是多少？

- | |
|---|
| <ol style="list-style-type: none">1、$p+1 \Leftrightarrow a[1]$2、$*p++=1 \Leftrightarrow a[2]=1, p=\&a[3]$3、$p_2 - p_1 = 3$ |
|---|



练习：

1、 `int a[]={1,2,3,4,5,6}, *p=a, i;`则数组元素地址的正确表示：

(A) `&(a+1)` (B) `a++` (C) `&p` (D) `&p[i]`



`p++`: 合法，因为`p`是指针变量，`++`只能用于变量。

`a++`: 不合法，因为`a`是数组名，其值是数组元素的首地址，是一个地址常量，因此`a`的值不能改变。

2、找出下面程序中的错误并改正

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int i, a[3], *p;
7      p = a;
8
9      for(i=0; i<3; i++, p++)
10         cin >> *p;
11
12     for(i=0; i<3; i++, p++)
13         cout << *p << " ";
14
15     return 0;
16 }
```

```
1 2 3
0 2293312 0
```



```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int i, a[3], *p;
7      p = a;
8
9      for(i=0; i<3; i++, p++)
10         cin >> *p;
11
12     for(i=0, p=a; i<3; i++, p++)
13         cout << *p << " ";
14
15     return 0;
16 }
```

```
1 2 3
1 2 3
```

从上例可知，指针变量p可以指向数组以后的内存单元，编译系统不作检查。

6.3.3 一级指针变量与一维数组的关系

已知 `int a[10], *p; p=a;` 则需注意以下问题：

- 数组名是地址常量；
- `p+i` 是 `a[i]` 的地址；
- 数组元素的引用有下标法和指针法： $p[i] \Leftrightarrow q[i] \Leftrightarrow *(p+i) \Leftrightarrow *(q+i)$ ；
- 形参数组本质上是指针变量，即：形参 `int q[]` \Leftrightarrow `int *q`；
- 在定义指针变量（不是形参）时，不能把 `int *p` 写成 `int p[]`；
- 系统只给 `p` 分配能保存一个指针值的内存区（如2字节），而给 `a` 分配 2×10 字节的内存区。

6.4 指针与字符串

□ 字符串的引用方式

可以通过字符数组或字符串变量或指针变量三种方式引用字符串。

6.4.1 用字符数组处理字符串

```
char string[] = "I love China!";  cout << string << endl;
```

6.4.2 用字符串变量处理字符串

```
string str = "I love China!";  cout << str << endl;
```

6.4.3 用字符指针处理字符串

```
char *str = "I love China!";  cout << str << endl;
```

6.5 指针与函数

6.5.1 指向函数的指针变量

在编译时，编译系统为函数代码分配一段存储空间，这段存储空间的起始地址（又称为入口地址）称为这个**函数的指针**。

□ 定义形式

数据类型 (*指针变量名)(函数参数表列);

说明：

- 数据类型是指函数返回值的数据类型；
- “指针变量”存放函数的入口地址，可指向同类型的不同函数；
- “(*指针变量名)”中的()不能省，`int (*p)()`和`int *p()`是不同的。
- 对于指向函数的指针变量：`p±n`，`p++`，`p--`无意义。

□ 指向函数的指针变量的初始化

已知 `int max(int, int), (*p)(int, int);`

则有 `p = max;`

□ 函数调用方式

已知 `int max(int, int), (*p)(int, int); p = max`

则 `max` 函数的调用方式有以下两种：

- `c = max(a, b);`
- `c = (*p)(a, b);`

例6.2 求a和b中的大者，要求用指向函数的指针变量实现程序。

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int max(int, int);
7      int (*p)(int, int);
8      int a,b,c;
9      p = max; //不能写成 p = max(a, b)
10     cin >> a >> b;
11     c = (*p)(a, b); //通过指向max函数的指针变量p调用max函数
12     cout << "a = " << a << " b = " << b << " max = " << c;
13     return 0;
14 }
15
16 int max(int x,int y)
17 {
18     int z;
19     if(x>y) z=x;
20     else z=y;
21     return(z);
22 }
```

```
12 56
a = 12 b = 56 max = 56
```


6.5.2 返回指针值的函数

□ 定义形式

类型名 *函数名(参数表列)

说明：

- 函数返回值是指针类型，而该指针指向的目标变量类型函数定义中的“类型名”给定；
- 注意区分 `int (*p)()` 与 `int *p()`。

6.5.3 指针作为函数参数

指针作为函数的参数实现了地址的传递，从而使得形参与实参指向同一个存储单元，实现了存储单元的共享和数据的双向传递。

□ 指向变量的指针变量作为函数参数

□ 指向数组元素的指针变量作为函数参数

6.5.3.1 指向变量的指针变量作为函数参数

比较以下程序，分析指针变量作为函数参数的特点

例6.3.1

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int x, int y)
5  {
6      int temp;
7
8      temp = x; //值的交换
9      x = y;
10     y = temp;
11 }
12
13 int main()
14 {
15     int a, b;
16     cin >> a >> b;
17     if(a<b) swap(a, b); //值传递
18     cout << "a = " << a << " b = " << b << endl;
19     return 0;
20 }
```

3 5

a = 3, b = 5

例6.3.2

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int *p1, int *p2)
5  {
6      int temp;
7      temp = *p1; //对p1和p2指向的存储单元中的值进行交换
8      *p1 = *p2;
9      *p2 = temp;
10 }
11
12 int main()
13 {
14     int a, b;
15     cin >> a >> b;
16     int *pointer_1, *pointer_2;
17     pointer_1 = &a;
18     pointer_2 = &b;
19     if(a<b) swap(pointer_1, pointer_2); //地址传递
20     cout << "a = " << a << " b = " << b << endl;
21     return 0;
22 }
```

3 5

a = 5, b = 3

例6.3.3

```
1 #include <iostream>
2 using namespace std;
3
4 void swap(int *p1, int *p2)
5 {
6     int *temp;
7     *temp = *p1; // 对p1和p2指向的存储单元中的值进行交换
8     *p1 = *p2;
9     *p2 = *temp;
10 }
11
12 int main()
13 {
14     int a, b;
15     cin >> a >> b;
16     int *pointer_1, *pointer_2;
17     pointer_1 = &a;
18     pointer_2 = &b;
19     if(a < b) swap(pointer_1, pointer_2); // 地址传递
20     cout << "a = " << a << " b = " << b << endl;
21     return 0;
22 }
```



例6.3.4

```
1 #include <iostream>
2 using namespace std;
3
4 void swap(int *p1, int *p2)
5 {
6     int *temp;
7     temp = p1; // 对p1和p2值进行交换
8     p1 = p2;
9     p2 = temp;
10 }
11
12 int main()
13 {
14     int a, b;
15     cin >> a >> b;
16     int *pointer_1, *pointer_2;
17     pointer_1 = &a;
18     pointer_2 = &b;
19     if(a < b) swap(pointer_1, pointer_2); // 地址传递
20     cout << "a = " << a << " b = " << b << endl;
21     return 0;
22 }
```

```
3 5
a = 3, b = 5
```

□ 指向数组元素的指针变量作为函数参数

若有如下定义及初始化：

```
int a[10], *p1 = a;
```

```
int b[3][4], *p2 = b[0];
```

则实参和形参对应形式如下表所示：

	实参	形参
一维数组	数组名a	数组名int x[]
	数组名a	一级指针变量int *q
	一级指针变量p1	一级指针变量int *q
	一级指针变量p1	数组名int x[]

6.6 指针数组和多级指针

6.6.1 指针数组

□ 指针数组是指数组元素均为指针变量的数组。

□ 定义形式：

类型名 *数组名[数组长度]；

- 说明：1、指针数组的定义：`int *p[4];`
指向一维数组的指针变量的定义：`int (*p)[4]`。

2、类型名是指针指向的目标变量的类型。

□ 指针数组的初始化和赋值

举例1：

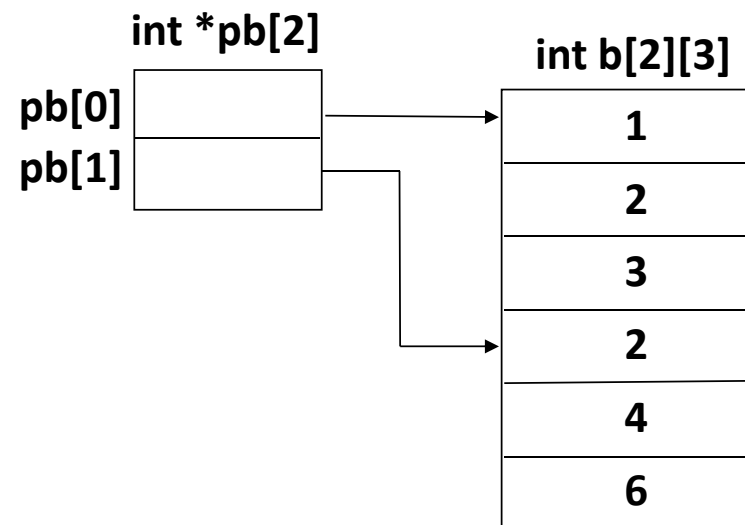
```
int b[2][3], *pb[ ]={b[0], b[1]};
```

或

```
int b[2][3], *pb[2];
```

```
pb[0] = b[0];
```

```
pb[1] = b[1];
```



举例2：

```
char *p[]={ "Fortran", "Lisp", "Basic", NULL};
```

或

```
char a[]="Fortran";
```

```
char b[]="Lisp";
```

```
char c[]="Basic";
```

```
char *p[4];
```

```
p[0]=a; p[1]=b; p[2]=c; p[3]=NULL;
```

或

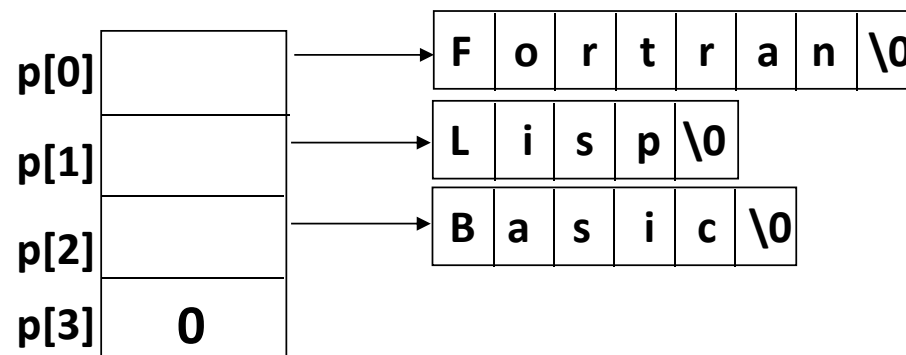
```
char *p[4];
```

```
p[0]= "Fortran";
```

```
p[1]= "Lisp";
```

```
p[2]= "Basic";
```

```
p[3]=NULL;
```



例6.4 对字符串排序（简单选择排序）

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  int main()
6  {
7      void sort(char *name[],int n);
8      void print(char *name[],int n);
9
10     char *name[]={"Follow me", "BASIC", "Great Wall",
11                  "FORTRAN", "Computer design"};
12     int n = 5;
13     sort(name, n);
14     print(name, n);
15
16     return 0;
17 }
```

```
BASIC
Computer design
FORTRAN
Follow me
Great Wall
```

```
19 void sort(char *name[], int n)
20 {
21     char *temp;
22     int i, j, k;
23     for(i=0; i<n-1; i++) {
24         k = i;
25         for(j=i+1; j<n; j++)
26             if(strcmp(name[k],name[j]) > 0)
27                 k = j;
28
29         if(k!=i) {
30             temp = name[i];
31             name[i] = name[k];
32             name[k] = temp;
33         }
34     }
35 }
```

```
37 void print(char *name[], int n)
38 {
39     int i ;
40     for(i=0; i<n; i++)
41         cout << name[i] << endl;
42 }
```

6.6.2 多级指针

□ 多级指针是指向指针数据的指针，简称指向指针的指针。

□ 二级指针变量的定义形式

数据类型 **指针名；

• 说明：

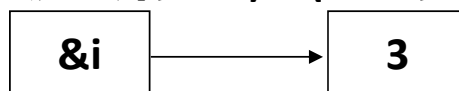
- 1、数据类型是指最终目标变量的数据类型。
- 2、例如：char **p，其中：*p是p间接指向对象的地址，而**p是p间接指向对象的值。
- 3、三级指针变量定义 int ***p;
四级指针变量定义 char ****p;

□ 二级指针变量的理解

- 一级指针变量：指针变量中存放目标变量的地址。

例如 `int *p;`
`int i = 3;`
`p = &i;`
`*p = 5;`

p(一级指针变量) i(整型变量)



一级指针

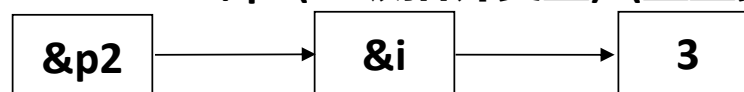
目标变量

一级
间接寻址

- 二级指针变量：指针变量中存放一级指针变量的地址。

例如 `int **p1;`
`int *p2;`
`int i = 3;`
`p2 = &i;`
`p1 = &p2;`
`**p1 = 5;`

p1(二级指针变量) p2(一级指针变量) i(整型变量)



二级指针

一级指针

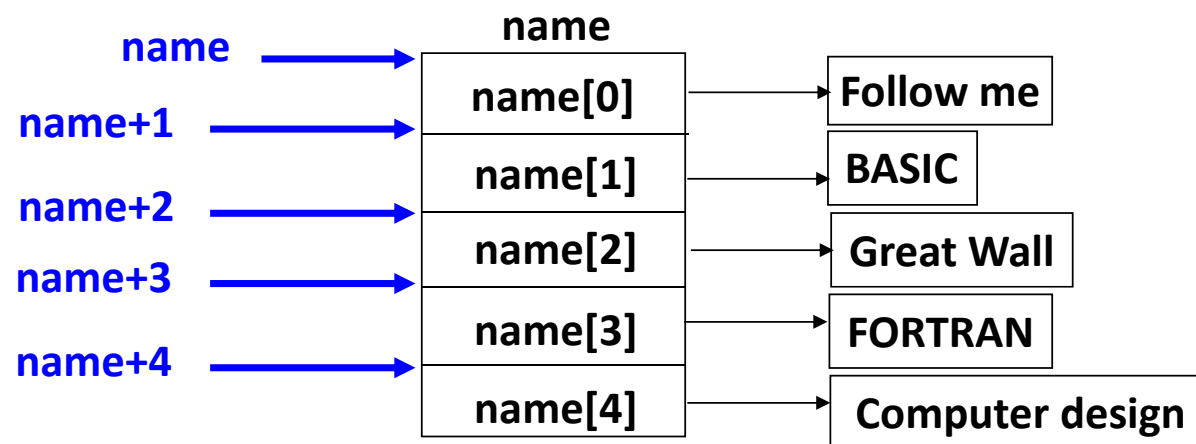
目标变量

二级
间接寻址

例6.5 用指向指针的指针处理字符串

```
1  #include <iostream>
2  using namespace std;
3
4  int main( )
5  {
6      char *name[]={ "Follow me", "BASIC",
7                      "Great Wall", "FORTRAN", "Computer design" };
8      char **p;    //p是二级指针变量
9      int i;
10
11     for(i=0; i<5; i++) {
12         p = name+i; //p指向第i个字符串
13         cout << *p << endl; // *p是name[i]的值, 即第i个字符串的起始地址
14     }
15
16     return 0;
17 }
```

```
Follow me
BASIC
Great Wall
FORTRAN
Computer design
```



□ 二级指针与指针数组的关系

已知 `int **p` 与 `int *q[10]` , 则 :

- 指针数组名是二级指针常量 ;
- 若 `p = q`; 则 `p+i` 是 `q[i]` 的地址 ;
- 若指针数组作形参 , 则 `int *q[]` 与 `int **q` 完全等价。但作为变量定义两者不同 ;
- 系统只给 `p` 分配能保存一个指针值的存储空间 ; 而给 `q` 分配10块连续的存储空间 , 每块空间存储一个指针值。

6.6.3 指针数组作为main函数的形参

指针数组的一个重要应用是作为main函数的形参。在以往的程序中，main函数的头部一般是无参函数形式，即int main()。

实际上，main函数可以有参数。有参main函数的定义如下：

```
int main(int argc, char *argv[])
```

其中，int argc和char *argv[]是main函数的形参，其具体含义如下：

- int argc：命令行中的命令和所有参数的个数之和。
- char *argv[]：指针数组，分别指向各个参数字符串的首地址。

□ 有参main函数的调用

main函数是由操作系统调用的。因此，有参main函数的实参是以命令行方式运行C程序的可执行文件（.exe）时，在命令行中给出的。也就是在一个命令行中包括命令名（.exe）和需要传给有参main函数的实参。

注：命令行是指在命令提示符中，为执行某个程序而键入的一行字符。

- 命令行的一般形式如下：

命令名 参数1 参数2 参数n

例如：C:\TC> copy[.exe] source.c temp.c

- 命令行参数传递

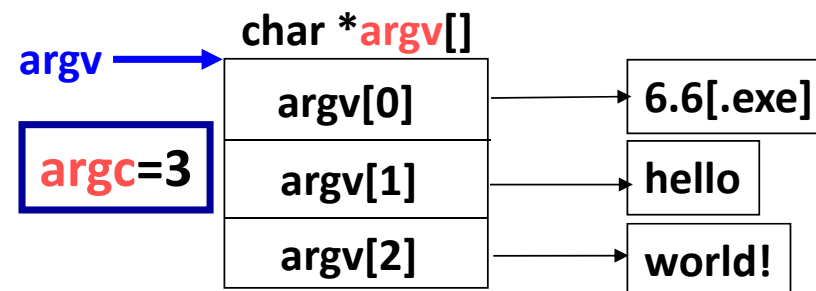


例6.6 用命令行方式运行程序——有参main函数的调用

```
1  #include <iostream>
2  using namespace std;
3
4  int main(int argc, char *argv[])
5  {
6      while(argc-->0)
7          cout << *argv++ << endl;
8
9      return 0;
10 }
```

1. 对程序（xx.cpp）进行编译、链接，生成可执行文件（xx.exe）；
2. 在命令提示符中输入命令名（xx[.exe]）和若干实参，并运行。

```
C:\Users\vc\Desktop\C++\程序\CH06>6.6.exe hello world!
6.6.exe
hello
world!
```



6.7 void类型指针和NULL指针

□ void类型指针

- void类型指针 (void *) 是一种通用指针类型，可以指向任意数据类型数据的指针变量。
- 定义形式

`void *p;`

说明：1、指向具体数据类型数据时，必须进行强制类型转换；
2、不能进行指针运算，也不能进行间接引用。

- 举例

若 `char *p1; void *p2;` 则 `p1 = (char *)p2;` 或 `p2 = (void *)p1;`

□ NULL指针

- **NULL是一个值，一个指针值，任何类型的指针都可以赋予该值。**
例如：`int *p; p = NULL;` 或 `char *c; c = NULL;`
- **在C/C++中，NULL通常被定义为一种预处理宏，其取值为0或 `(void *)0`，用以表示该指针不指向任何对象。**
- **用途：1) 避免指针变量的非法引用；2) 在程序中常被用作进行状态比较。**

例如：`int *p;`

`while(p != NULL)`

`{ ... }`

6.8 C++的动态内存分配new和delete

new和delete是运算符，不是函数

□ new运算符

形式1：new 类型说明符

分配一个能够存储该类型变量的内存空间，返回指向这个内存空间的指针

形式2：new 类型说明符[长度]

分配一个能够存储该类型数组的内存空间，返回指向这个内存空间的指针

例如：int *p=new int;

int *p=new int(5); //分配一个存储int型变量的内存空间，初始化为5，并将指向这个内存空间的指针保存在指针变量p中

int *p=new int[5];

6.8 C++的动态内存分配new和delete

new和delete是运算符，不是函数

□ new运算符

形式1：new 类型说明符

分配一个能够存储该类型变量的内存空间，返回指向这个内存空间的指针

形式2：new 类型说明符[长度]

分配一个能够存储该类型数组的内存空间，返回指向这个内存空间的指针

说明：1、new运算符返回一个与new所分配对象类型相匹配的指针；2、如果new运算符不能分配到所需要的内存，将返回NULL指针。

例如：`int *p=new int;`

`int *p=new int(5);` //分配一个int型大小的内存空间，并初始化该空间的指针为int型，
//设置初始值为5，然后将这个int型指针保存在指针变量p中

`int *p=new int[5];`

◆ **注意：**用new开辟的内存空间没有名字，指向它的指针是引用这片空间的唯一途径，若指针变量重新赋值，则用new开辟的内存空间就在系统中“丢失”了。

□ delete运算符

delete运算符用来释放new分配到的内存空间

形式1：delete 指针变量

如：int *p = new int; delete p;

形式2：delete [] 指针变量

如：int *p = new int[5]; delete [] p;

说明：1、必须用于由new返回的指针（包括NULL指针）；

2、一个指针只能delete一次。

6.9 const指针

6.9.1 指向常量的指针变量

const 类型名 *指针变量名

- 不能通过这种指针修改它指向的目标变量的值。

6.9.2 常指针变量

类型名 * const 指针变量名

- 这种指针变量称为常指针变量，简称常指针，即指针的值不能修改；
- 必须在定义时初始化；
- 这种指针指向的目标变量的值可以修改。

6.10 指针的类型和指针运算小结

□ 指针的类型

定 义	含 义
<code>int i ;</code>	定义整型变量i
<code>int *p ;</code>	定义p为指向整型数据的指针变量
<code>int a[n] ;</code>	定义由n个元素组成的整型数组a
<code>int *p[n] ;</code>	定义由n个指向整型数据的指针元素组成的指针数组p
<code>int (*p)[n] ;</code>	定义p为指向含n个元素的一维数组的指针变量
<code>int f() ;</code>	定义f为返回整型值的函数
<code>int *p() ;</code>	定义p为返回一个指针值的函数，该指针指向整型数据
<code>int (*p)() ;</code>	定义p为指向函数的指针，该函数返回一个整型值
<code>int **p ;</code>	定义p为一个指针变量，它指向一个指向整型数据的指针变量

□ 指针运算小结

(1) 指针变量加（减）一个整数，加减的值的大小与目标变量的类型有关

如：p++；p--；p+i；p-i；p+=i；p-=i等。

(2) 指针变量赋值

p=&a（将变量a的地址赋给p）

p=array；（将数组array首地址赋给p）

p=&array[i]；（将数组array第i个元素的地址赋给p）

p=max；（max为已定义的函数，将max函数的入口地址赋给p）

p1=p2；（p1和p2都是指针变量，将p2的值赋给p1）

(3) 指针变量若不指向任何变量，即取空值，表示为：p = NULL；

(4) 两个指针变量可以相减

如果两个指针变量指向同一个数组为元素，则两个指针变量值之差是两个指针之间的元素个数，但p1+p2并无实际意义。

(5) 两个指针变量比较

如果两个指针变量指向同一个数组为元素，则可以进行地址比较。

***6.11 引用**

给变量起一个别名，这个别名就是该变量的引用。

6.10.1 引用的定义和初始化

□ 引用的定义

类型 &引用名 = 变量名

其中，变量名必须是一个已经定义过的变量的名字。

例如：`int max; int &ref_max = max;`

这里，ref_max没有被分配存储单元，只是引用了max的存储单元，它们具有相同的内存地址，即一个存储单元两个名字。

□ 引用的初始化和赋值

- 定义引用时必须初始化；
- 可以用引用给变量赋值；
- 引用可以被赋值。

例如：`int a=3; int &m=a;` //定义引用并初始化，引用m是变量a的别名

`int n=m;` //用引用m给变量n赋值，实质是用变量a给变量n赋值

`int *p=&m;` //取引用m的地址（即变量a的地址）给指针p赋值，此时p指向变量a

`m=m+5;` //`a=a+5` → `a=8`，对引用的操作就是对被引用的变量的操作

说明：

- 1、引用作为函数形参时，它的初始化是在函数调用时实现的，是对实参的引用。**
- 2、引用一旦被初始化后就不能作为其他变量的别名；**
- 3、不能建立void类型的引用，不能建立引用的数组，不能建立引用的引用；**

□ 引用与指针的区别

- 1、指针是通过地址间接访问某个变量，而引用是通过别名直接访问某个变量；**
- 2、引用必须初始化，而一旦被初始化后不得再作为其它变量的别名。**

6.11.2 引用与函数

引用主要是用来做函数的形参或函数的返回值类型。

□ 引用作为函数形参

实质上是在被调用函数中对实参变量进行操作。

例6.7

```
1  #include <iostream>
2  using namespace std;
3
4  void swap(int &x, int &y)
5  {
6      int temp;
7      temp = x;
8      x = y;
9      y = temp;
10 }
11
12 int main()
13 {
14     int a, b;
15     cin >> a >> b;
16     swap(a, b);
17     cout << a << " " << b << endl;
18     return 0;
19 }
```

引用作为形参，实参是变量而不是变量的地址，这与指针变量作为形参是不一样的！

```
3 7
7 3
```

□ 引用作为函数的返回值类型

可以把函数定义为引用类型，这时函数的返回值即为某一变量的引用（别名），因此，它相当于返回了一个变量，所以可对其返回值进行赋值操作。但这个变量必须是全局变量或静态局部变量，而不能是自动变量和形参。

例6.8

```
1  #include <iostream>
2  using namespace std;
3
4  int a = 4;
5  int &f(int x)
6  {
7      a = a+x;
8      return a;
9  }
10
11 int main(void)
12 {
13     int t = 5;
14     cout << f(t) << endl;
15     f(t) = 20;
16     cout << "a=" << a << " f(t)=" << f(t) << endl;
17     t = f(t);
18     cout << "f(t)=" << f(t) << " t=" << t << endl;
19     return 0;
20 }
```

```
9
a=25 f(t)=25
f(t)=60 t=30
```