

5

系统级编程初探

5.1 引言

5.2 指针的基本概念与用法

5.3 函数中的指针

5.4 指针用于内存操作

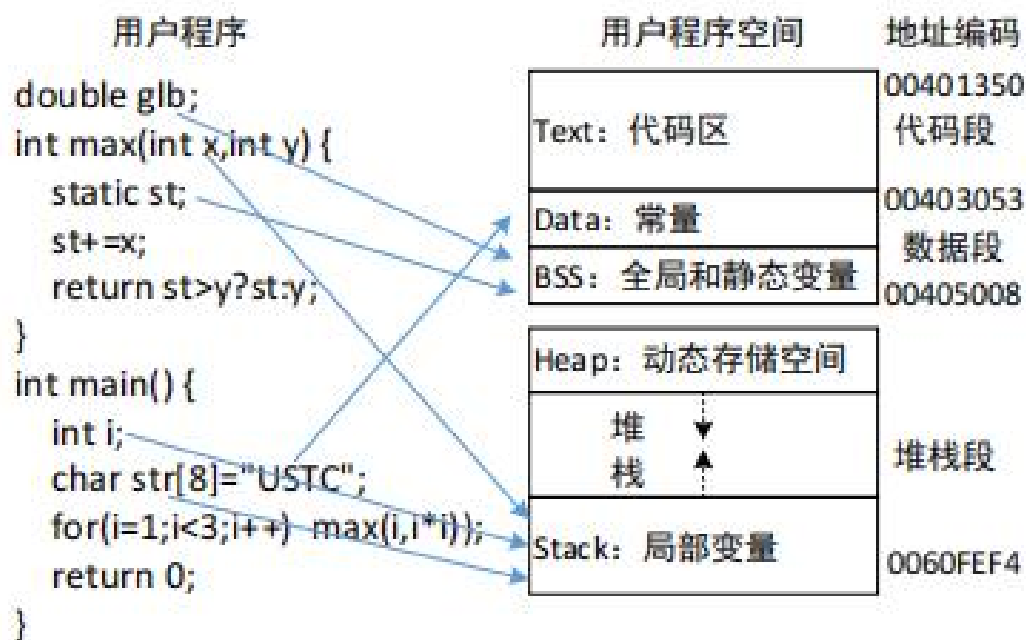
5.5 小结



- C 语言在设计之初就被用于编写操作系统，能面向系统进行编程是 C 语言流行至今的最重要原因。指针是使用 C 语言进行系统级编程时最强有力的工具，也是 C 语言中最具特色的精华所在。
- 本章以指针为基础，阐述程序在内存中对数组、函数、结构体、文件等进行操作时最底层的机制，揭示程序运行与数据处理的最基本的原理。



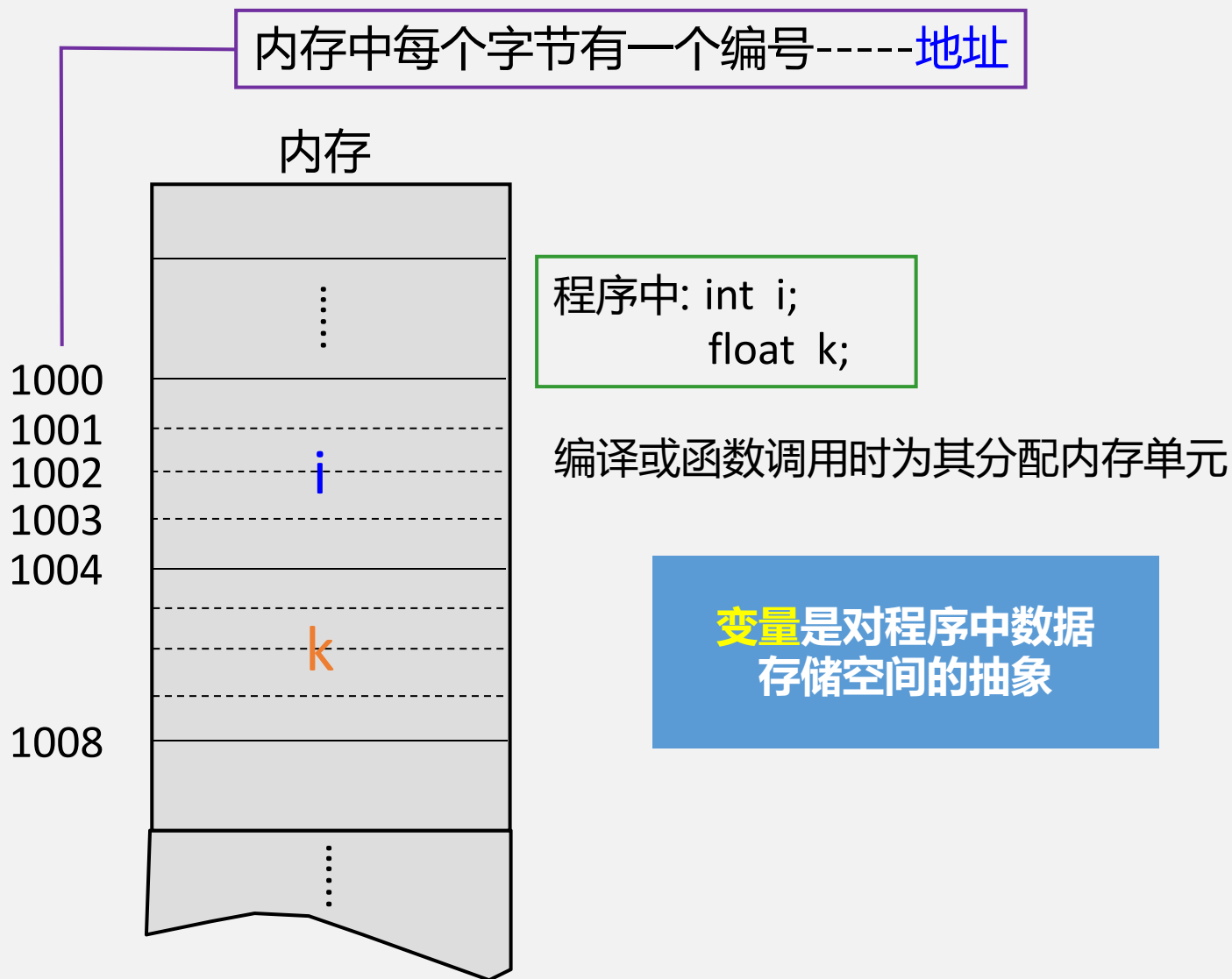
- 现代操作系统将内存分为内核空间和用户空间





指针的概念

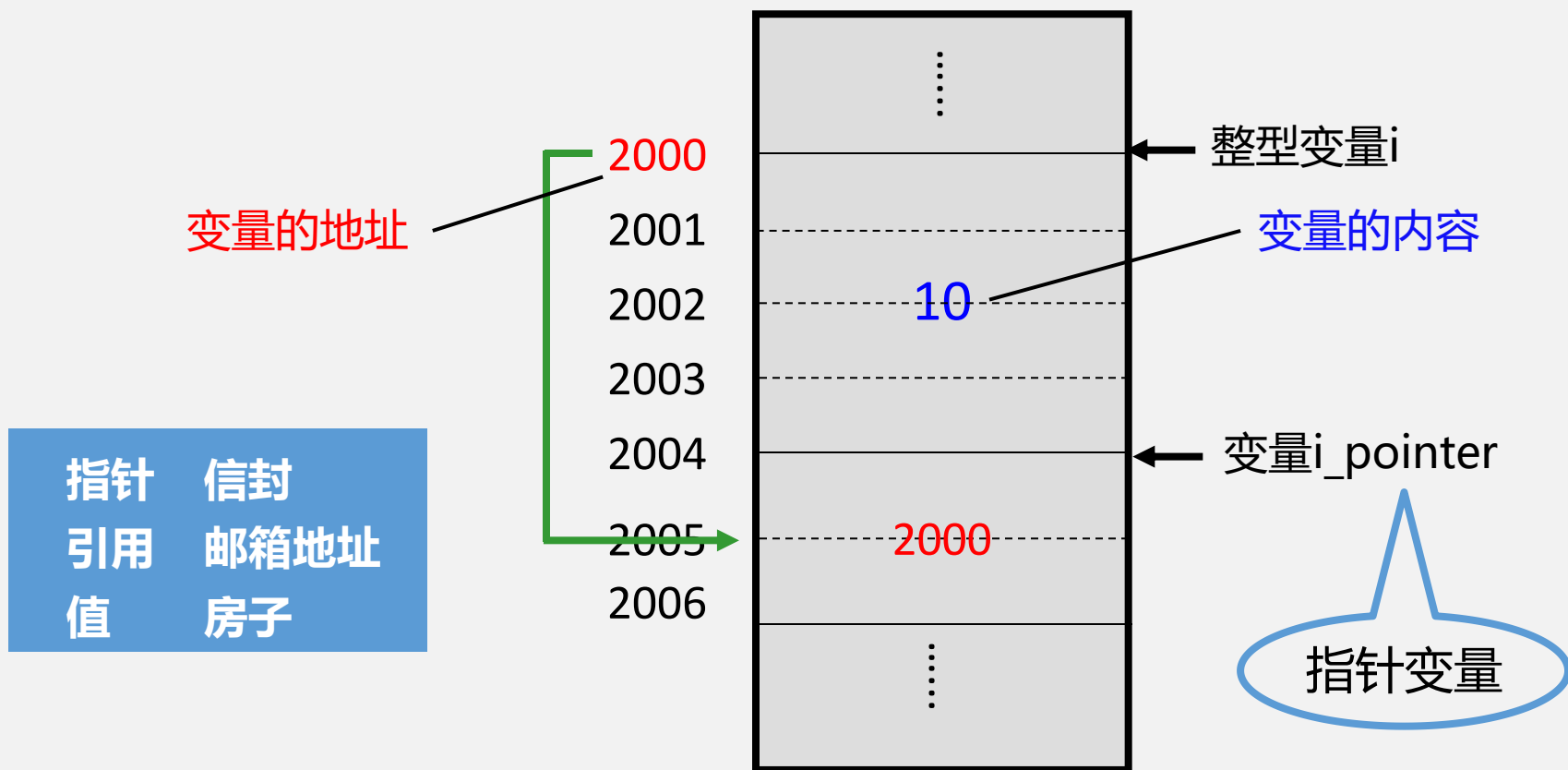
变量与地址





指针的概念

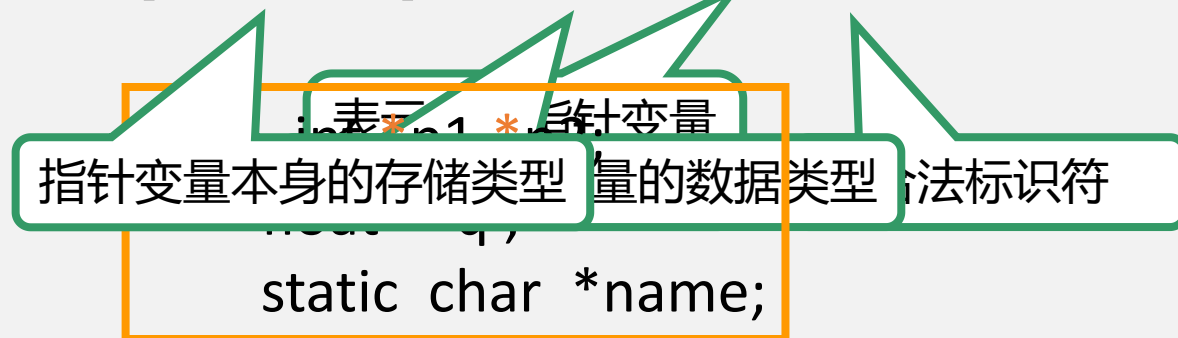
指针变量：专门存放变量地址的变量叫指针变量





指针的定义

❖ 一般形式： [存储类型] 数据类型 *指针名;



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



指针变量的初始化

一般形式：[存储类型] 数据类型 *指针名=初始地址值(NULL);

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



赋给指针变量，
不是赋给目标变量

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

变量必须已定义过
类型应一致

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

用已初始化指针变量作初值



指针变量的引用

&

含义: 取变量的地址

单目运算符

优先级: 2

结合性: 自右向左

*

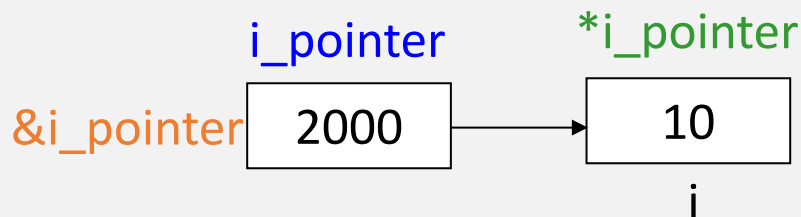
含义: 取指针所指向的变量

单目运算符

优先级: 2

结合性: 自右向左

❖ 两者关系: 互为逆运算



```
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

2000

2001

2002

2003

2004

2005

2006

整型变量*i*变量*i_pointer*

指针变量


10

2000



指针变量的引用

指针变量必须 **先赋值,再使用**

```
例  main( )  
    {  int  i=10;  
        int  *p;   
        *p=i;  
        printf("%d",*p);  
    }
```

```
例  main( )  
    {  int  i=10,k;  
        int  *p;  
        p=&k;  
        *p=i;  
        printf("%d",*p);  
    }
```



指针变量的引用

指针变量必须 初始化



指针变量的引用

空指针（零指针）

```
#define NULL 0
```

```
int *p=NULL;
```

表示指针变量的值为空
不指向任何变量或函数

注意：

- “#define NULL 0”已在stdio.h中定义
- 除0以外的任何整数都不允许直接赋给指针变量
- 定义指针时将其初始化为NULL可以避免指向未知区域
- **定义指针未指向有效对象时保持NULL是良好的习惯**

无效指针（invalid pointer）

- 定义指针变量后未赋值
- 将整型变量转换成指针
- 释放指针所指对象的存储空间
- 指针运算超出范围



指针变量的引用

例 5.1 输入两个数，并使其从大到小输出

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

```
int main()
```

```
{
```

```
    int *p1=NULL,*p2=NULL,*p=NULL;
```

```
    int a=3,b=5;
```

```
    p1=&a; p2=&b;
```

```
    if(a<b)
```

```
    { p=p1; p1=p2; p2=p; }
```

```
    printf("a=%d,b=%d\n",a,b);
```

```
    printf("max=%d,min=%d\n",*p1,*p2);
```

```
    return 0;
```

```
}
```

指针的定义

指针的赋值

指针的交换

指针的引用



指针变量的引用

例 5.2 按正向和反向顺序打印一个字符串

```
#include<stdio.h>
int main()
{
    char *ptr1=NULL,*ptr2=NULL;
    ptr1="happy new year";
    ptr2=ptr1;
    while(*ptr2!='\0')
        putchar(*ptr2++);
    putchar('\n');
    while(--ptr2>=ptr1)
        putchar(*ptr2);
    putchar('\n');
    return 0;
}
```

***ptr2++的运算**



指向数组的指针

数组名是表示数组首地址的地址常量

```
int array[10];
```

```
int *p=array;
```



```
int array[10];
```

```
int *p=&array[0];
```



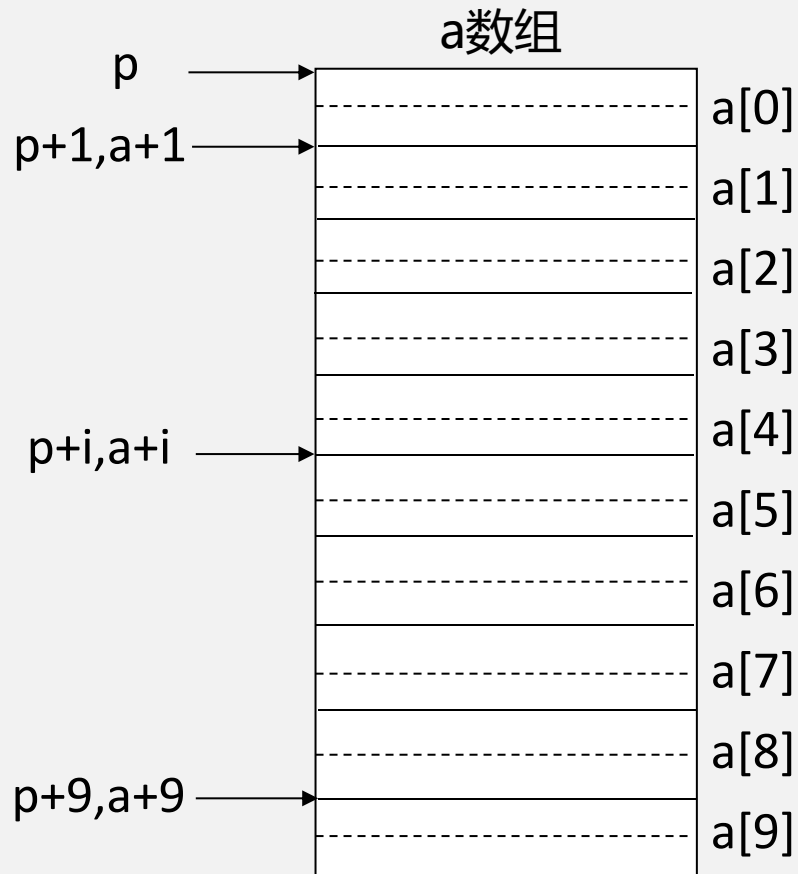
指向数组的指针

```
int a[10];
```

```
int *p=a;
```

使用a访问数组

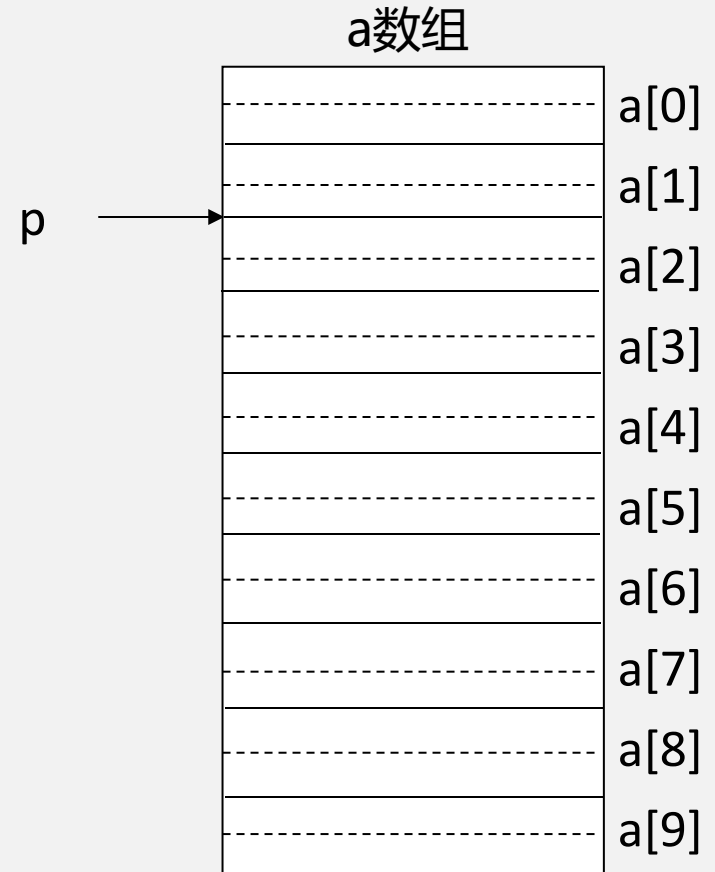
使用p访问数组





指向数组的指针

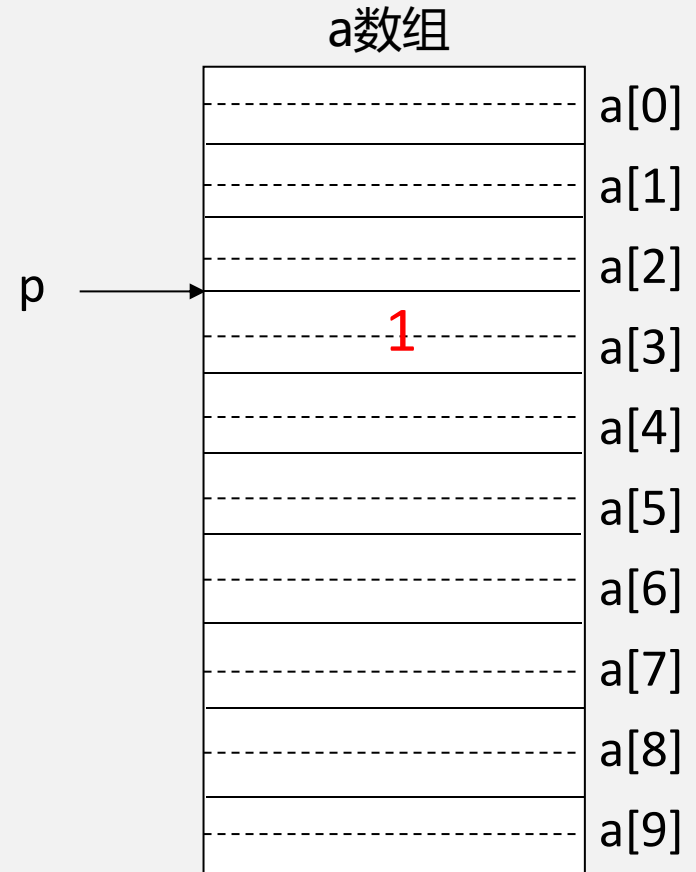
```
int a[10];  
int *p=&a[2];  
p++;  
*p=1;
```





指向数组的指针

```
int a[10];  
int *p=&a[2];  
p++;  
*p=1;
```



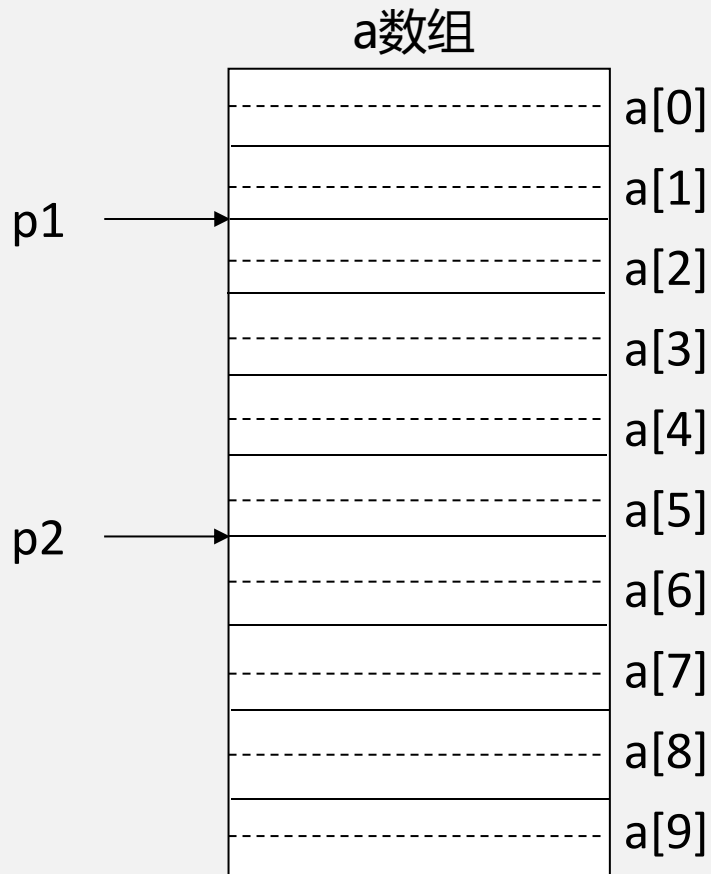


指向数组的指针

```
int a[10];  
int p1=&a[2];  
int p2=&a[6];
```



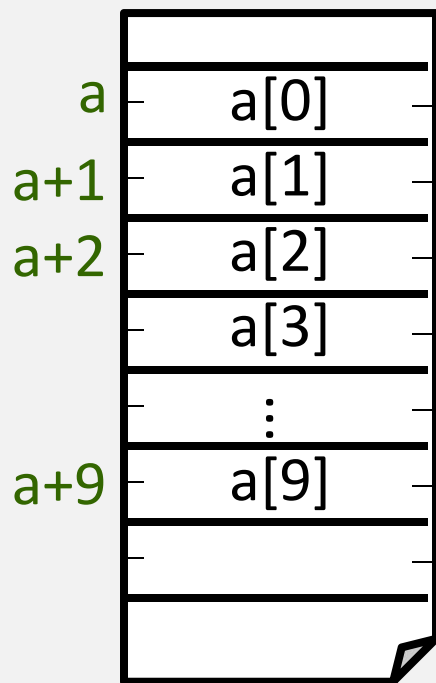
```
p2-p1=4;
```





指向数组的指针

地址



下标法

元素

a[0] *a
a[1] *(a+1)
a[2] *(a+2)
⋮
a[9] *(a+9)

地址

p
p+1
p+2
⋮
p+9

元素

*p p[0]
*(p+1) p[1]
*(p+2) p[2]
⋮
*(p+9) p[9]

指针法

[] 变址运算符
 $a[i] \Leftrightarrow *(a+i)$

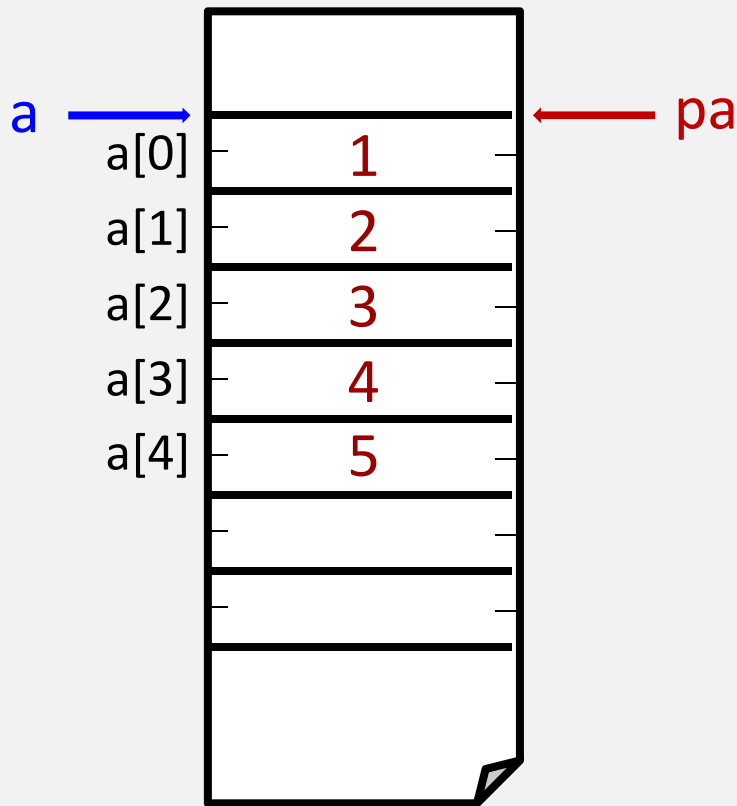
$a[i] \Leftrightarrow p[i] \Leftrightarrow *(p+i) \Leftrightarrow *(a+i)$



指针变量的引用

例 5.3 用数组名和指针引用数组元素

```
#include<stdio.h>
int main()
{
    int a[5]={1,2,3,4,5},i;
    int *pa=a;
    for(i=0;i<5;i++)
        printf("%d\t",*(a+i));putchar('\n');
    for(i=0;i<5;i++)
        printf("%d\t",*(pa+i));putchar('\n');
    for(i=0;i<5;i++)
        printf("%d\t",pa[i]);putchar('\n');
    for(i=0;i<5;i++)
        printf("%d\t",a[i]);putchar('\n');
    for(i=0;i<5;i++)
        printf("%d\t",*pa++);putchar('\n');
    return 0;
}
```





指针变量的引用

例 5.3 用数组名和指针引用数组元素

`pa++`

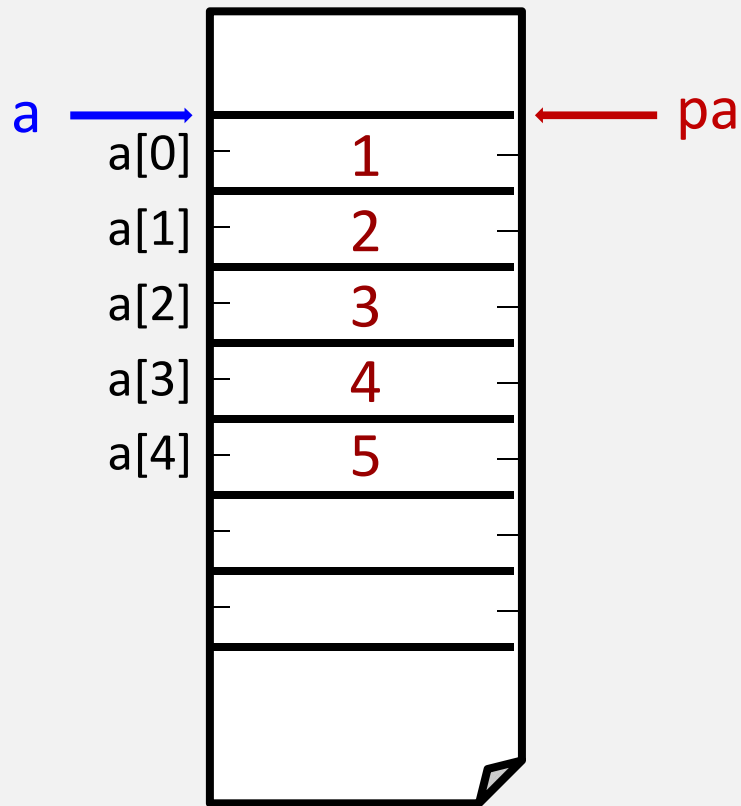
`++pa`

`*pa++`

`(*pa)++`

`*(pa++)`

`*++pa`





指针变量的引用

例 5.4 编写字符串复制函数函数udf_strcpy()

```
int main()
{
    char s[15]="abc",t[]="def";
    udf_strcpy4(s,t);
    puts(s);
    return 0;
}
```

```
void udf_strcpy1(char s[],char t[]){
    int i=0;
    while((s[i]=t[i])!='\0')
        i++;
}
```

```
void udf_strcpy2(char *s,char *t){
    while((*s=*t)!='\0'){
        s++; t++;
    }
}
```

```
void udf_strcpy3(char *s,char *t){
    while((*s++=*t++)!='\0');
}
```

```
void udf_strcpy4(char *s,char *t){
    while(*s++=*t++);
}
```



指针变量的引用

例 5.5 反置数组 (递归)

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void invert(char *s,int i,int j);
int udf_strlen(char *s);
int main()
{
    char str[80];
    int i=0,j;
    gets(str);puts(str);
    j=udf_strlen(str);
    printf("%d\n",j);
    invert(str,i,j-1);
    puts(str);
    return 0;
}
```

```
void invert(char *s,int i,int j){
    char t;
    if(i<j){
        t=*(s+i);
        *(s+i)=*(s+j);
        *(s+j)=t;
        invert(s,i+1,j-1);
    }
}
```

```
int udf_strlen(char *s){
    int i=0;
    while(*s++) i++;
    return(i);
}
```


例5-6 如有int a[]={1,2,3,4,5,6,7,8,9,10},*p=a,i;
则数组元素地址的正确表示为：

- ☐ A &(a+1)
- ☐ B a++
- ☐ C &p
- ☒ D &p[i]

提交

例5-7 有以下程序

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

```
int main()
```

```
{
```

```
    int a[]={5,8,7,6,2,7,3};
```

```
    int y,*p=&a[1];
```

```
    y=(*--p)++;
```

```
    printf("%d ",y);
```

```
    printf("%d",a[0]);
```

```
    return 0;
```

```
}
```

输出结果为

v= [填空1] *p= [填空2] a[0]= [填空3]

正常使用填空题需3.0以上版本雨课堂

作答



指针变量的引用

例 5.8 指针越界

```
#include<stdio.h>
int main()
{
    int i,*p=NULL,a[2];
    p=a;
    for(i=0;i<2;i++){
        printf("a[%d]:",i);
        scanf("%d",p++);
    }
    printf("\n");
    for(i=0;i<2;i++,p++)
        printf("a[%d]=%d\n",i,*p);
    return 0;
}
```

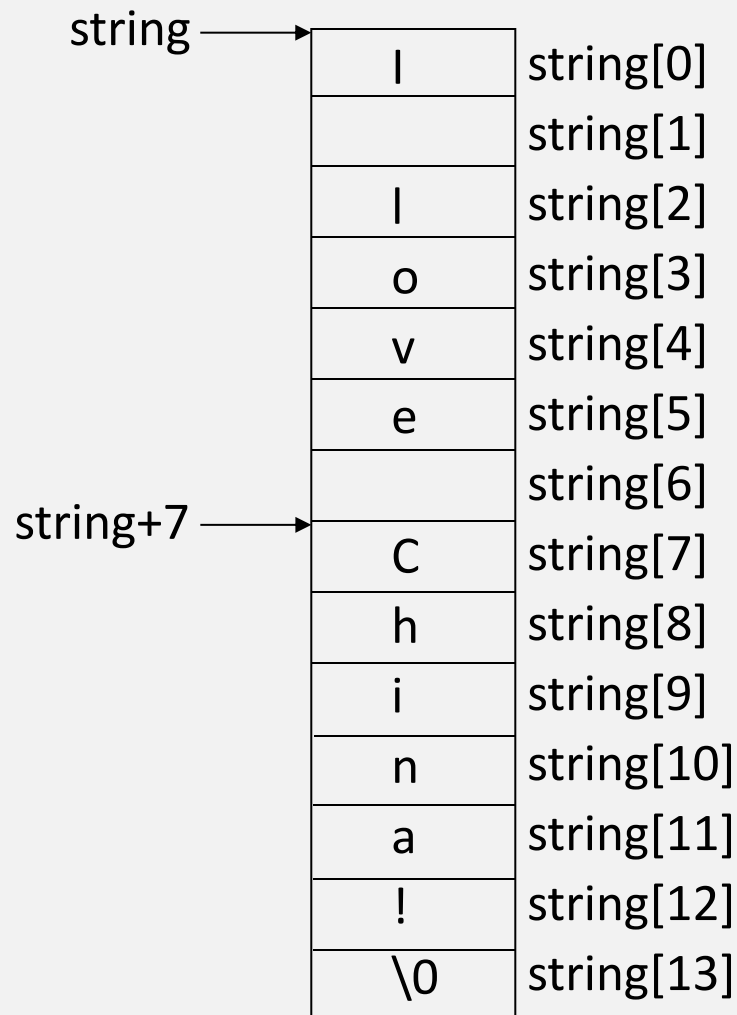
- 由于 C 语言不做数组越界检查，在程序中需要注意避免指针越过数组的地址范围。



字符数组与指针

```
#include <stdio.h>

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

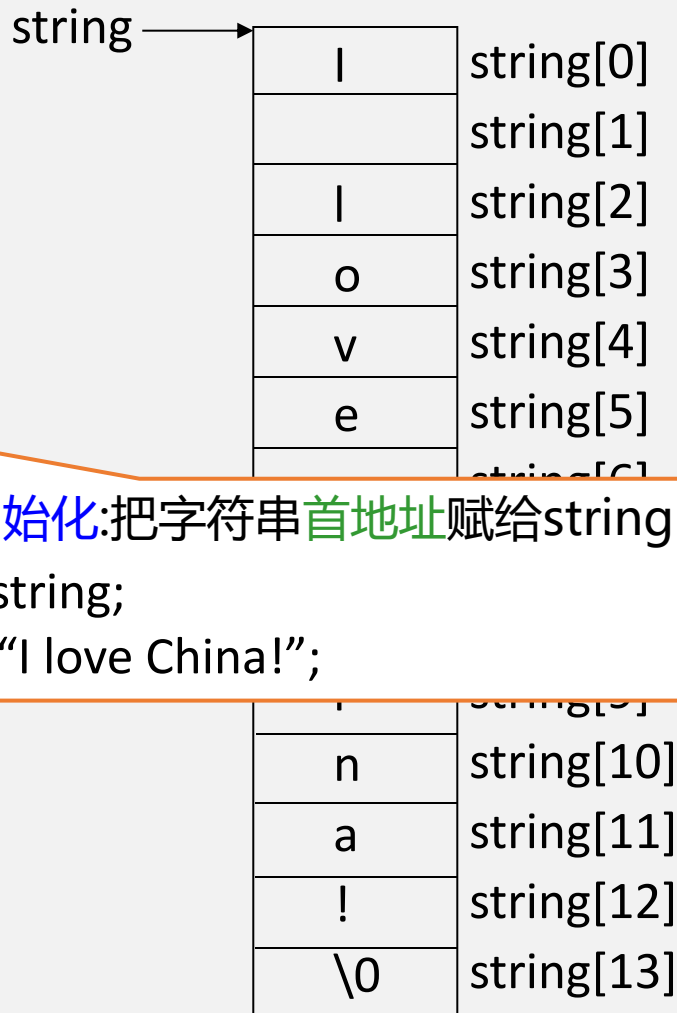




字符数组与指针

```
#include <stdio.h>

int main( )
{
    char *string="I love China!";
    printf("%s\n",string);
    string+=7;
    while(*string){
        putchar(string[0]);
        string++;
    }
    return 0;
}
```



字符指针初始化:把字符串首地址赋给string

⇔ char *string;
string="I love China!";



字符数组与指针

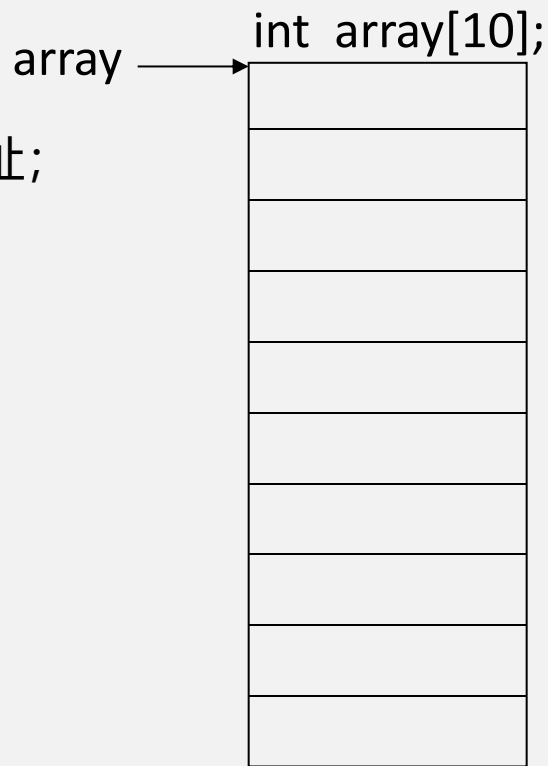
例 5.9 利用指向字符串的指针统计每个字母出现的次数

```
#include <stdio.h>
int main()
{
    char *str="The new Science ... way.";
    char c;
    int anum[26]={0};
    puts(str);
    while(c=*str++){
        if(c>='a'&&c<='z') anum[c-'a']++;
        if(c>='A'&&c<='Z') anum[c-'A']++;
    }
    printf("\n字母 a 和 A 有%d 个\n",anum[0]);
    printf("字母 b 和 B 有%d 个\n",anum[1]);
    return 0;
}
```



对于一维数组:

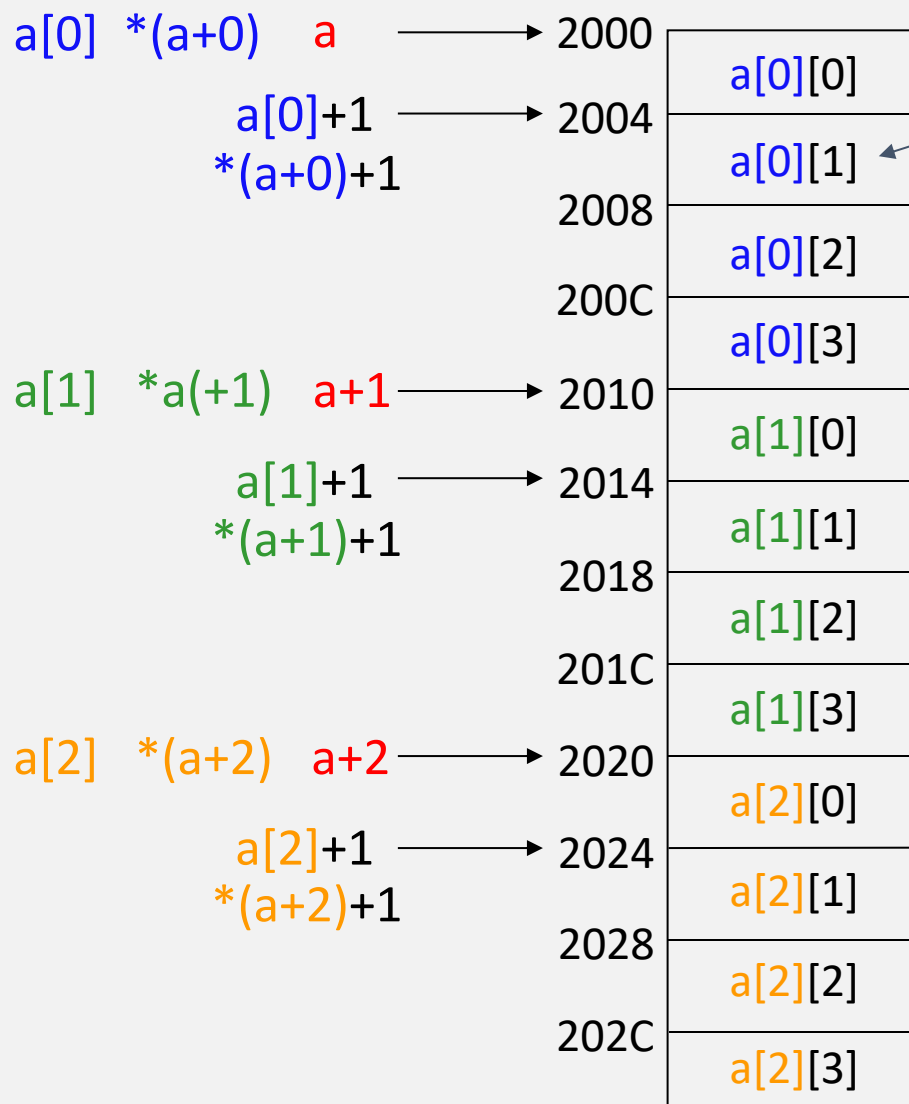
- 1、数组名array表示数组的首地址，即array[0]的地址；
- 2、数组名array是地址常量
- 3、array+i是元素array[i]的地址
- 4、array[i] \Leftrightarrow *(array+i)





二维数组与指针

行指针和列指针

`int a[3][4];``*(a[0]+1)``*(*(a+0)+1)`

对于二维数组:

(1) `a`、`a+1`、`a+2`是行地址`a[0]`、`a[1]`、`a[2]`是列地址(2) `a[0]`、`a[1]`、`a[2]`等同于
`*(a+0)`、`*(a+1)`、`*(a+2)`



二维数组与指针

对二维数组 `int a[3][4]`, 有 (从第0行第0列算起)

- ❖ `a`-----二维数组的首地址, 即第0行的首地址
- ❖ `a+i`-----第*i*行的首地址
- ❖ `a[i] ⇔ *(a+i)`-----第*i*行第0列的元素地址
- ❖ `a[i]+j ⇔ *(a+i)+j`-----第*i*行第*j*列的元素地址
- ❖ `*(a[i]+j) ⇔ *(*a+i)+j ⇔ a[i][j]`

`a+i = &a[i] = a[i] = *(a+i) = &a[i][0]`, 地址编号相等, 含义不同

`a+i ⇔ &a[i]`, 表示第*i*行首地址, 指向行

`a[i] ⇔ *(a+i) ⇔ &a[i][0]`, 表示第*i*行第0列元素地址, 指向列

`int a[3][4];`

<code>a[0][0]</code>
<code>a[0][1]</code>
<code>a[0][2]</code>
<code>a[0][3]</code>
<code>a[1][0]</code>
<code>a[1][1]</code>
<code>a[1][2]</code>
<code>a[1][3]</code>
<code>a[2][0]</code>
<code>a[2][1]</code>
<code>a[2][2]</code>
<code>a[2][3]</code>



二维数组与指针

地址表示:

- (1) $a+1$ ← 行地址
 - (2) $\&a[1][0]$
 - (3) $a[1]$
 - (4) $*(a+1)$
- } 列地址

地址表示:

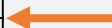
- (1) $\&a[1][2]$
- (2) $a[1]+2$
- (3) $*(a+1)+2$
- (4) $\&a[0][0]+1*4+2$

二维数组元素表示形式:

- (1) $a[1][2]$
- (2) $*(a[1]+2)$
- (3) $*(*(a+1)+2)$
- (4) $*(&a[0][0]+1*4+2)$

`int a[3][4];`

<code>a[0][0]</code>
<code>a[0][1]</code>
<code>a[0][2]</code>
<code>a[0][3]</code>
<code>a[1][0]</code>
<code>a[1][1]</code>
<code>a[1][2]</code>
<code>a[1][3]</code>
<code>a[2][0]</code>
<code>a[2][1]</code>
<code>a[2][2]</code>
<code>a[2][3]</code>



例5-10 有以下程序

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

```
int main()
```

```
{
```

```
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
```

```
    int *p=NULL;
```

```
    for(p=a[0];p<a[0]+12;p++){
```

```
        if((p-a[0])%4==0) printf("\n");
```

```
        printf("%4d",*p);
```

```
    }
```

```
    return 0;
```

```
}
```

在代码蓝色部分作如下替换哪个是错误的?

- ☐ A p=*a;
- ☐ B p=&a[0][0];
- ☐ C p=*(a+0);
- ☒ D p=a;

提交



二维数组与指针

指向一维数组的指针变量 (**行指针**)

❖ 定义形式 **数据类型** (***指针名**) [**一维数组长度**];

p的值是一维数组的首地址，p是**行指针**

例 `int (*p)[4];`

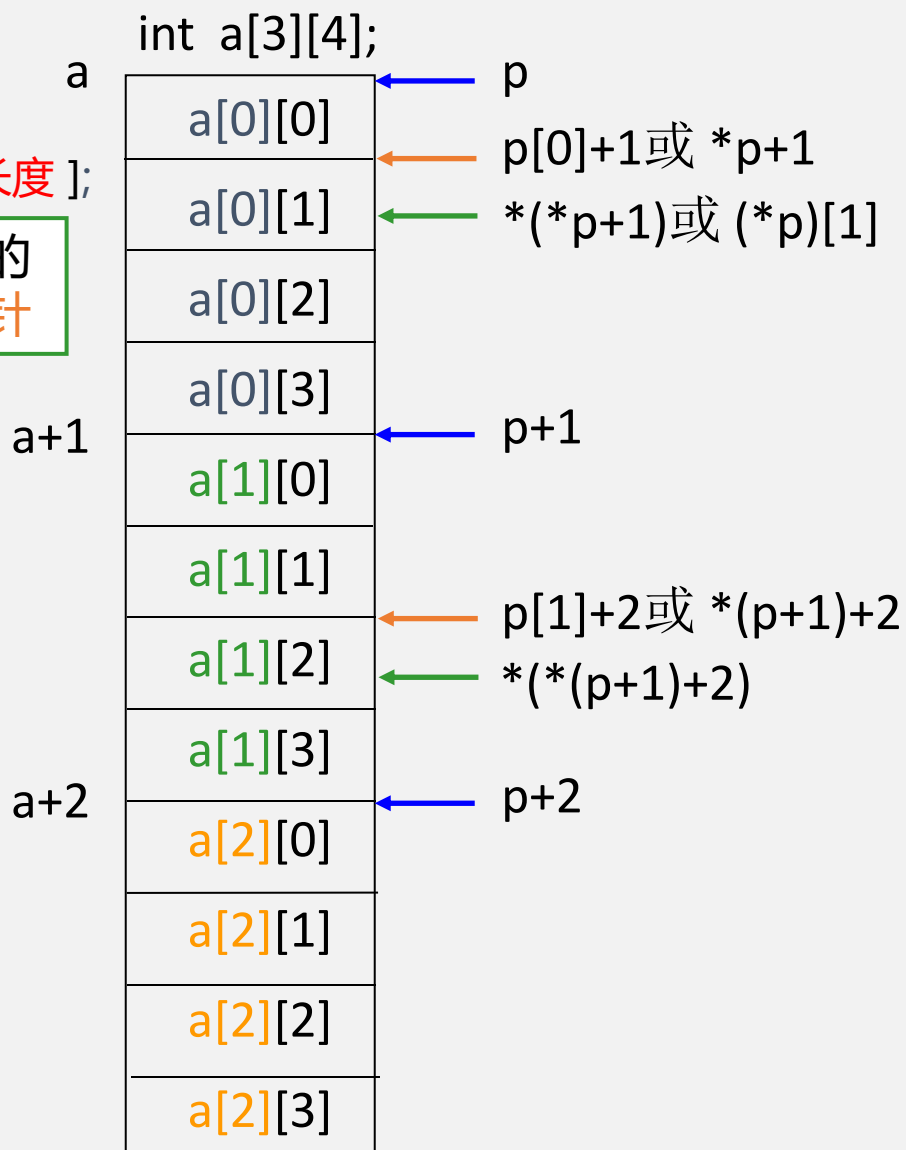
()不能少

`int (*p)[4]`与`int *p[4]`不同

❖ 可让p指向二维数组某一行

如 `int a[3][4], (*p)[4]=a;`

一维数组指针变量长度和二维数组列数**必须相同**



例5-11 有以下程序

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

```
int main()
```

```
{
```

```
    int a[3][4]={1,3,5,7,9,11,13,15,17,19,21,23};
```

```
    int i,j,(*p)[4]=NULL;
```

```
    for(p=a,i=0;i<3;i++,p++)
```

```
        for(j=0;j<4;j++)
```

```
            printf("%d ",*(*p+j));
```

```
    printf("\n");
```

```
    return 0;
```

```
}
```

在代码蓝色部分作如下替换哪个是正确的?

☐ A p=a[0]

☒ B p=&a[0]

☐ C p=*a

☐ D p=&a[0][0]

提交

填空题 2分



例5-12 有以下程序

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

```
int main()
```

```
{
```

```
    int a[3][4]={1,2,3,4},{3,4,5,6},{5,6,7,8}};
```

```
    int i,j;
```

```
    int (*p)[4]=a,*q=a[0];
```

```
    for(i=0;i<3;i++){
```

```
        if(i%2==0)
```

```
            (*p)[i/2+1]=*q+3;
```

```
        else
```

```
            p++,++q;
```

```
    }
```

```
    for(i=0;i<3;i++){
```

```
        for(j=0;j<4;j++)
```

```
            printf("%4d",a[i][j]);
```

```
            putchar('\n');
```

```
    }
```

```
    printf("%d,%d\n",**p,*q);
```

```
    return 0;
```

```
}
```

输出结果为

****p= [填空1] ,*q= [填空2]**

正常使用填空题需3.0以上版本雨课堂

作答



二维数组与指针

- ❖ 二维数组的指针做函数参数
 - 用指向变量的指针变量
 - 用指向一维数组的指针变量
 - 用二维数组名

若 `int a[3][4]; int (*p1)[4]=a; int *p2=a[0];`

实参	形参
数组名a	数组名 <code>int x[][4]</code>
数组名a	指针变量 <code>int (*q)[4]</code>
指针变量p1	数组名 <code>int x[][4]</code>
指针变量p1	指针变量 <code>int (*q)[4]</code>
指针变量p2	指针变量 <code>int *q</code>



二维数组与指针

例 5.13 3个学生各学4门课，计算总平均分，并输出第n个学生成绩

```
#include <stdio.h>
int main()
{
    void average(float *p,int n);
    void search(float (*p)[4],int n);
    float score[3][4]=
    {{65,67,79,60},{80,87,90,81},
     {90,99,100,98}};
    average(*score,12);
    search(score,2);
    return 0;
}
```

```
void average(float *p,int n){
    float *p_end=NULL, sum=0,aver;
    p_end=p+n-1;
    for(;p<=p_end;p++)
        sum=sum+(*p);
    aver=sum/n;
    printf("average=%5.2f\n",aver);
}
```

```
void search(float (*p)[4], int n){
    int i;
    printf(" No.%d :\n",n);
    for(i=0;i<4;i++)
        printf("%5.2f ",*(*(p+n)+i));
}
```




二维数组与指针

例 5.14 3个学生3门课，对平均分进行排序

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
struct stu{
    int num;
    int score_1;
    int score_2;
    int score_3;
    float ave;
};
int main()
{
    void RandomIntArray(struct stu *p_array,int num);
    void Sort(struct stu *p_array,int num);
    void Print(struct stu *p_array,int num);
    struct stu student[3];
    RandomIntArray(student,3);
    Print(student,3);
    Sort(student,3);
    printf("\nsorted:\n");
    Print(student,3);
    return 0;
}
```

```
void RandomIntArray(struct stu *p_array,int num)
{
    int i=1;
    srand(time(NULL)+rand());
    while(num--){
        (*p_array).num=i++;
        (*p_array).score_1 = rand() % 100;
        (*p_array).score_2 = rand() % 100;
        (*p_array).score_3 = rand() % 100;
        (*p_array).ave=((*p_array).score_1+(*p_array).score_2+(*p_array).score_3)/3.0;
        p_array++;
    }
}

void Print(struct stu *p_array,int num)
{
    int i;
    for(i=0;i<3;i++,p_array++)
        printf("student%d : score1 %d\tscore2 %d\tscore3 %d\taverage%f\n",(*p_array).num,score_2,(*p_array).score_1,(*p_array).score_2,(*p_array).score_3,(*p_array).ave);
}

void Sort(struct stu *p_array,int num)
{
    int i,j;
    struct stu temp;
    for(i=0;i<num-1;i++)
        for(j=0;j<num-1-i;j++)
            if(p_array[j].ave>p_array[j+1].ave)
                {temp=p_array[j];p_array[j]=p_array[j+1];p_array[j+1]=temp;}
}
```



指针数组

- ❖ 定义：数组中的元素为指针变量
- ❖ 定义形式：[存储类型] 数据类型 *数组名[数组长度];

指针本身的存储类型

指针所指向变量的数据类型

例 static int *p[4];

区分int *p[4]与int (*p)[4]



指针数组

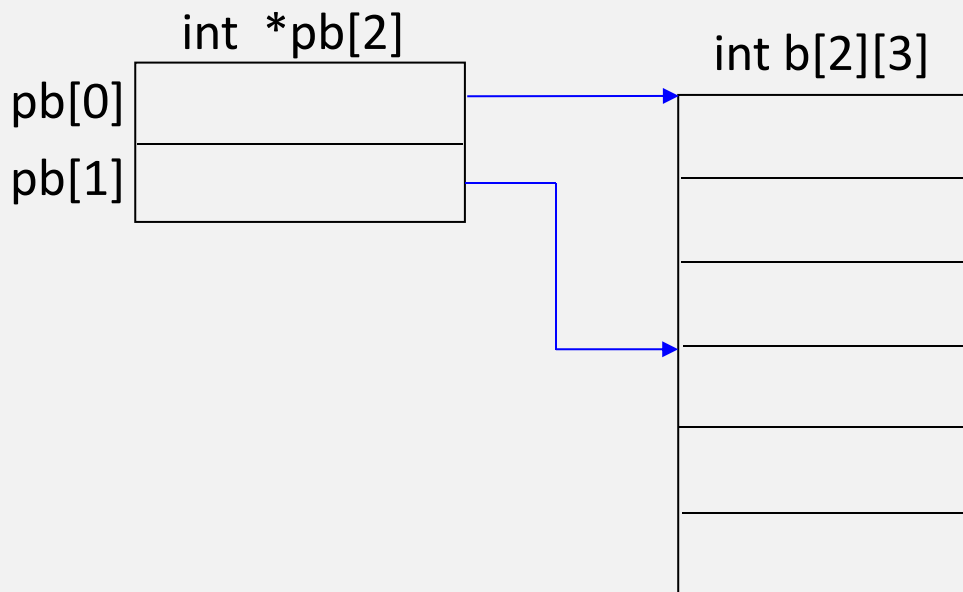
❖ 指针变量的赋值与初始化

赋值:

```
int main()
{  int b[2][3], *pb[2]={NULL};
    pb[0]=b[0];
    pb[1]=b[1];
    .....
}
```

初始化:

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





指针数组和字符串

❖ 二维字符串数组与指向字符串的指针数组区别：

```
char name[5][9]={“gain”,“much”,“stronger”, “point”,“bye”};
```

```
char *name[5]={“gain”,“much”,“stronger”, “point”,“bye”};
```

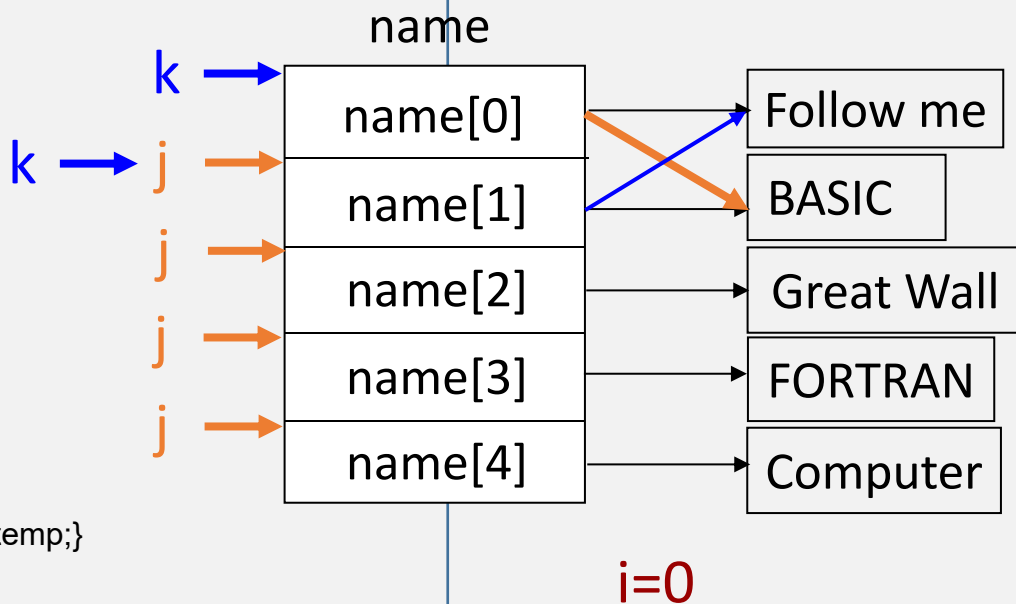
- 二维数组字符串数组存储空间固定，指向字符串的指针数组相当于每行长度不同的二维数组。
- 指向字符串的指针数组元素的作用相当于二维数组的行名，但指针数组中元素是指针变量，二维数组行名是地址常量



指针数组和字符串

例 5.15 3个学生3门课，对平均分进行排序

```
#include<stdio.h>
int main()
{
    void Sort(char *name[],int n)
    void Print(char *name[],int n);
    char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer "};
    int n=5;
    sort(name,n);
    print(name,n);
    return 0;
}
void sort(char *name[],int n)
{ char *temp=NULL;
  int i,j,k;
  for(i=0;i<n-1;i++){
      k=i;
      for(j=i+1;j<n;j++)
          if(strcmp(name[k],name[j])>0) k=j;
      if(k!=i)
          { temp=name[i]; name[i]=name[k]; name[k]=temp;}
  }
}
```

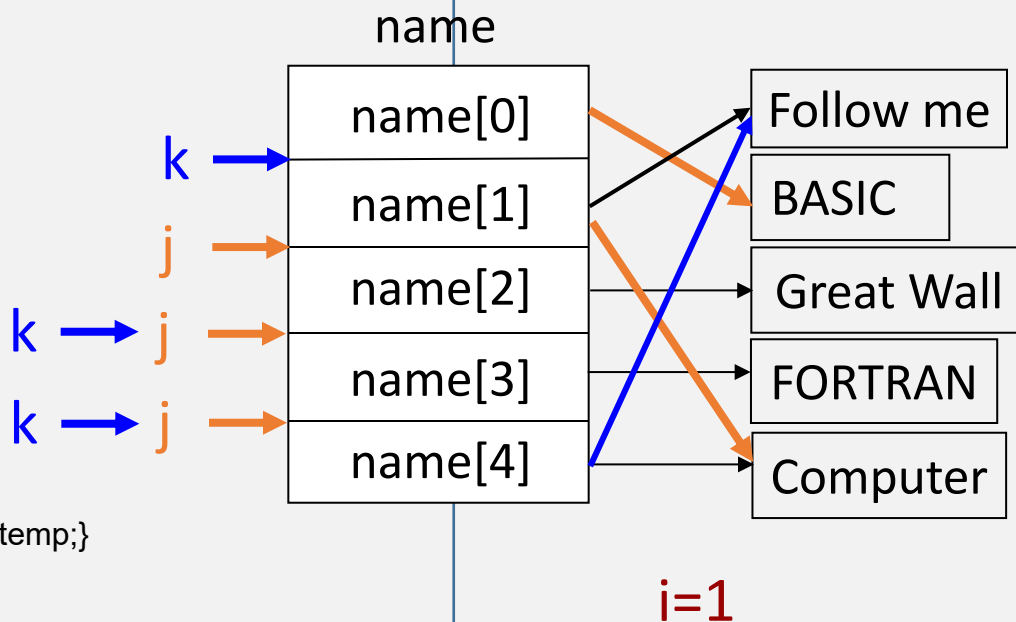




指针数组和字符串

例 5.15 3个学生3门课，对平均分进行排序

```
#include<stdio.h>
int main()
{
    void Sort(char *name[],int n)
    void Print(char *name[],int n);
    char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer "};
    int n=5;
    sort(name,n);
    print(name,n);
    return 0;
}
void sort(char *name[],int n)
{ char *temp=NULL;
  int i,j,k;
  for(i=0;i<n-1;i++){
      k=i;
      for(j=i+1;j<n;j++){
          if(strcmp(name[k],name[j])>0) k=j;
          if(k!=i)
          { temp=name[i]; name[i]=name[k]; name[k]=temp;}
      }
  }
}
```

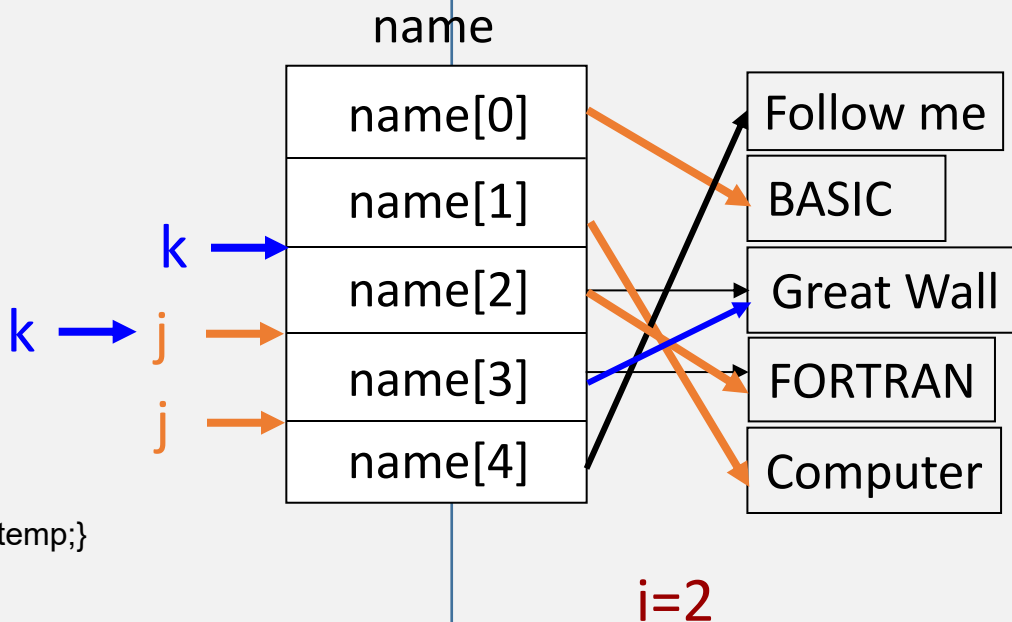




指针数组和字符串

例 5.15 3个学生3门课，对平均分进行排序

```
#include<stdio.h>
int main()
{
    void Sort(char *name[],int n)
    void Print(char *name[],int n);
    char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer "};
    int n=5;
    sort(name,n);
    print(name,n);
    return 0;
}
void sort(char *name[],int n)
{ char *temp=NULL;
  int i,j,k;
  for(i=0;i<n-1;i++){
      k=i;
      for(j=i+1;j<n;j++){
          if(strcmp(name[k],name[j])>0) k=j;
          if(k!=i)
          { temp=name[i]; name[i]=name[k]; name[k]=temp;}
      }
  }
}
```

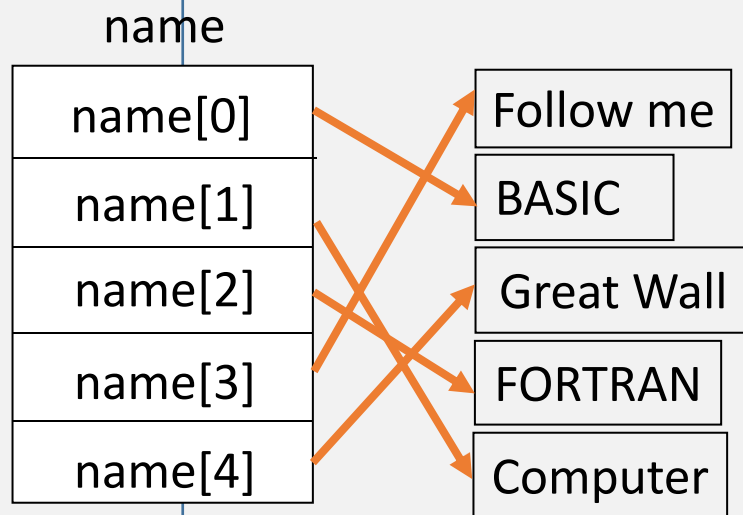




指针数组和字符串

例 5.15 3个学生3门课，对平均分进行排序

```
#include<stdio.h>
int main()
{
    void Sort(char *name[],int n)
    void Print(char *name[],int n);
    char *name[]={"Follow me","BASIC","Great Wall","FORTRAN","Computer "};
    int n=5;
    sort(name,n);
    print(name,n);
    return 0;
}
void sort(char *name[],int n)
{ char *temp=NULL;
  int i,j,k;
  for(i=0;i<n-1;i++){
      k=i;
      for(j=i+1;j<n;j++){
          if(strcmp(name[k],name[j])>0) k=j;
          if(k!=i)
          { temp=name[i]; name[i]=name[k]; name[k]=temp;}
      }
  }
}
```

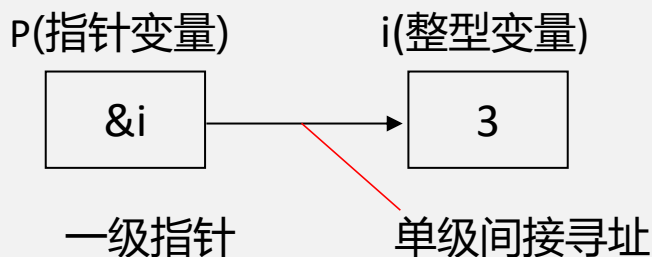




多级指针 (指向指针的指针)

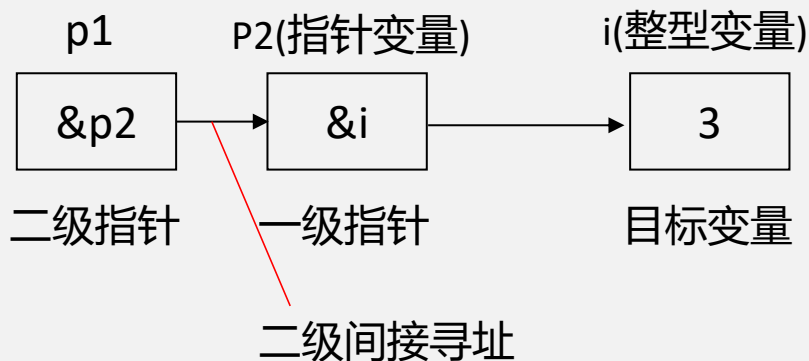
- ❖ 定义: 指向指针的指针
- ❖ 一级指针: 指针变量中存放目标变量的地址

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



- ❖ 二级指针: 指针变量中存放一级指针变量的地址

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





多级指针 (指向指针的指针)

最终目标变量的数据类型

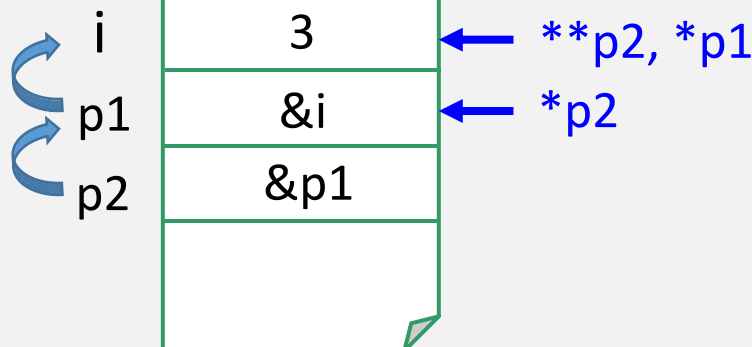
- 定义形式: [存储类型] 数据类型 **指针

如 char **p;

指针本身的存储类型

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

*p是p间接指向对象的地址
**p是p间接指向对象的值



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 三级指针 int ***p;

四级指针 char ****p;



多级指针和指针做函数参数

例 5.16 变量交换

```
#include <stdio.h>
void swap(int *r,int *s)
{
    int *temp=NULL;
    temp=r;
    r=s;
    s=temp;
}
int main()
{
    int a=1,b=2,*p=NULL,*q=NULL;
    p=&a;
    q=&b;
    swap(p,q);
    printf("%d,%d\n",*p,*q);
    return 0;
}
```

```
#include <stdio.h>
void swap(int **r,int **s)
{
    int *temp=NULL;
    temp=*r;
    *r=*s;
    *s=temp;
}
int main()
{
    int a=1,b=2,*p=NULL,*q=NULL;
    p=&a;
    q=&b;
    swap(&p,&q);
    printf("%d,%d\n",*p,*q);
    return 0;
}
```



多级指针和指针做函数参数

例 5.17 由用户设置浮点数输出的小数位数并输出。

```
#include <stdio.h>
int main()
{
    char format[]="%10.*f\n";
    double f=3.1415926535;
    int i;
    printf("number (0-9): ");
    scanf("%d", &i);
    format[4] = i + '0';
    printf(format,f);
    return 0;
}
```

? 能否用指针实现

用字符串首地址作为printf()函数的参数



多级指针和指针做函数参数

例 5.18 用指针数组和二级指针输出成绩

```
#include<stdio.h>
int main()
{
    int stu1[]={78,79,73,-1},stu2[]={100,98,-1},stu3[]={88,-1};
    int stu4[]={96,78,33,65,-1},stu5[]={99,88,-1};
    int *grade[]={stu1,stu2,stu3,stu4,stu5};
    int **p=grade,i;
    for (i=1;i<=5;i++){
        printf("student%d grade:",i);
        while(**p>=0){
            printf("%4d",**p);
            (*p)++;
        }
        p++;
        printf("\n");
    }
    return 0;
}
```

(p*)++和*p++*的不同含义?**



命令行参数

int main ()



有没有想过往这个括号里写点啥？



命令行参数

- ❖ 命令行：在操作系统状态下，为执行某个程序而键入的一行字符
- ❖ 命令行一般形式：**命令名** 参数1 参数2.....参数n

```
E:\> copy[.exe] source.c temp.c
```

有3个字符串参数的命令行

- ❖ 带参数的main函数形式：

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

命令行中参数个数

元素指向命令行参数
中各字符串首地址

注：1、形参名可以任意

2、形参char *argv[]本质上就是一个指向指针数组的**二级指针**，
所以通常也可以写成char **argv

- ❖ 命令行参数传递

系统自动调用
main函数时传递

第一个参数: main所在
的**可执行文件名**

命令行**实参**

main(**形参**)

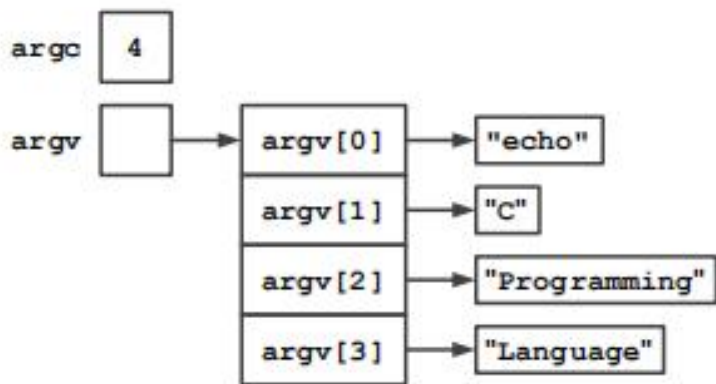


命令行参数

例 5.19 echo命令

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    while (--argc > 0)
        printf("%s%c", *++argv, (argc>1)? ' ':'\n');
    return 0;
}
```

循环输出字符串





命令行参数

例 5.20 用命令行参数实现两个实数之和

```
#include<stdio.h>
#include<process.h>
#include<math.h>
int main(int argc,char *argv[])
{
    double x,y;
    if(argc!=3){
        printf("using:command arg1,arg2<CR>\n");
        exit(1);
    }
    x=atof(argv[1]);
    y=atof(argv[2]);
    printf("sum=%f\n",x+y);
    return 0;
}
```

将字符串转为实数



指针函数

函数的类型被定义为指针类型，意味着它可以返回一个指针类型的返回值。

函数定义形式： 类型标识符 *函数名(参数表)例

例 `int *f(int x, int y)`



指针函数

例 5.21 用返回值和返回地址两种方法判断两个参数的大小

```
#include<stdio.h>
int larger1(int x,int y){return(x>y?x:y);}
int *larger2(int *x,int *y){return(*x>*y?x:y);}
int main()
{
    int a,b,bigger1,*bigger2=NULL;
    printf("input 1st integer values:");
    scanf("%d",&a);
    printf("input 2nd integer values:");
    scanf("%d",&b);
    bigger1=larger1(a,b);
    printf("the larger value is %d\n",bigger1);
    bigger2=larger2(&a,&b);
    printf("the larger value is %d\n",*bigger2);
    return 0;
}
```

传值返回值

传地址返回地址



指针函数

例 5.22 编写函数，生成整型随机数存放在动态分配的内存空间内，并返回地址

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
int *randdata(int n){
    int i;
    int *pdata = NULL;
    pdata = (int *)malloc(n * sizeof(int));
    if (pdata == NULL) {
        printf("Not enough memory!");
        exit(1);
    }
    srand((unsigned int)time(NULL));
    for (i=0; i<n; i++)
        *(pdata + i) = rand();
    return (pdata);
}
int main()
{
    int *p = NULL;
    int i;
    p = randdata(10);
    for (i=0; i<10; i++)
        printf("%5d\n", p[i]);
    return 0;
}
```

动态分配空间



指针函数

例 5.23 函数返回值为地址

```
#include<stdio.h>
int main()
{
    int *f(int,int);
    int a=2,b=3;
    int *p=NULL;
    p=f(a,b);
    printf("%d\n",*p);
    return 0;
}
int *f3(int x,int y)
{
    if(x>y) return &x;
    else    return &y;
}
```

注意： 不能将形参或局部变量的地址作为函数返回值



指向函数的指针

函数首地址：函数在编译时被分配的入口地址,用函数名表示

指向函数的指针变量

❖ 定义形式：数据类型 (*指针变量名)();

函数返回值的数据类型

()不能省

int (*p)() 与 int *p()不同

专门存放函数入口地址
可指向返回值类型不同的不同函数

❖ 函数指针变量赋值：如 `p=max;`

❖ 函数调用形式：`c=max(a,b);` \Leftrightarrow `c=(*p)(a,b);`

\Leftrightarrow `c=p(a,b);`

❖ 对函数指针变量`p±n`, `p++`, `p--`无意义



指向函数的指针

例 5.24 用函数指针变量调用函数，比较两个数大小

```
#include<stdio.h>
int main()
{
    int max(int,int),(*p)();
    int a,b,c;
    p=max;
    scanf("%d%d",&a,&b);
    c=(*p)(a,b);
    printf("a=%d,b=%d,max=%d\n",a,b,c);
    return 0;
}
int max(int x,int y){
    int z;
    if(x>y) z=x;
    else z=y;
    return(z);
}
```

这东西有啥用



指向函数的指针

例 5.25 用函数指针变量作参数，求最大值、最小值和两数之和

```
#include<stdio.h>
void main()
{ int a,b,max(int,int),
  min(int,int),add(int,int);
  void process(int,int,int (*fun)());
  scanf("%d,%d",&a,&b);
  process(a,b,max);
  process(a,b,min);
  process(a,b,add);
}
void process(int x,int y,int (*fun)())
{ int result;
  result=(*fun)(x,y);
  printf("%d\n",result);
}
```

```
max(int x,int y)
{ printf("max=");

return(x>y?x:y);
}
min(int x,int y)
{ printf("min=");

return(x<y?x:y);
}
add(int x,int y)
{ printf("sum=");
  return(x+y);
}
```



指向void量的指针变量

定义形式: **void** *合法标识符

```
void *xp;
```

- ❖ 其他类型指针 (char *, int * 等) 可以直接赋值给void指针
- ❖ void指针赋值给其他类型指针时, 必须进行**强制类型转换**

```
int vi1,vi2,*ip;  
float vf,*fp;  
void *xp;  
ip=&vi1;  
fp=&vf;  
ip=fp;      X  
fp=ip;      X  
xp=fp;      ✓  
xp=ip;      ✓  
ip=xp;      X  
ip=(int *)xp; ✓
```

- ❖ 当void型指针指向了具体的对象后, 需要用**强制类型转换**的方式读取对象内容

```
vi2=*(int *)xp;
```

注:

- ❖ void型指针在运算时, xp+1表示指针移动**一个字节**的距离
- ❖ void型指针经常用于编写通用函数



指向void量的指针变量

例 5.26 用void指针实现通用数据交换函数

```
#include<stdio.h>
void genswap(void *a,void *b,int size);
main()
{
    int m1=100,m2=200;
    double fx1=123.4,fx2=234.5;
    char str1[20]="Today is well day!";
    char str2[20]="今天是个好天气! ";
    genswap(&m1,&m2,sizeof(int));
    printf("m1=%d,m2=%d\n",m1,m2);
    genswap(&fx1,&fx2,sizeof(double));
    printf("fx1=%f,fx2=%f\n",fx1,fx2);
    genswap(str1,str2,sizeof(str1));
    printf("str1=%s,str2=%s\n",str1,str2);
}
void genswap(void *a,void *b,int size)
{
    char t;
    int i;
    for(i=0;i<size;i++){
        t=*((char *)a+i);
        *((char *)a+i)=*((char *)b+i);
        *((char *)b+i)=t;
    }
}
```

- 交换如何实现的?
- 为什么要将void指针强制转换成char?



指针的数据类型

定义	含义
<code>int i;</code>	定义整型变量i
<code>int *p;</code>	p为指向整型数据的指针变量
<code>int a[n];</code>	定义含n个整元素的整型数组a
<code>int *p[n];</code>	定义n个指向整型数据的指针变量组成的指针数组p
<code>int (*p)[n];</code>	定义指向含n个元素的一维整型数组的指针变量p
<code>int f();</code>	定义返回整型数的函数f
<code>int *p();</code>	定义返回值为指针的函数p，返回的指针指向一个整型数据
<code>int (*p)();</code>	定义指向函数的指针变量p，p指向的函数返回整型数
<code>int **p;</code>	定义二级指针变量p，它指向一个指向整型数据的指针变量



下列定义的含义

- `int *(*p)();` ← 指向函数的指针，函数返回int 型指针
- `int (*p[3])();` ← 函数指针数组，函数返回int型变量
- `int *(*p[3])();` ← 函数指针数组，函数返回int型指针

```
#include<stdio.h>
int *max(int *a,int *b){
    return(*a>*b?a:b);
}
int main()
{
    int *max(int*,int*);
    int *(*p)(),*pmax;
    int x=5,y=10;
    p=max;
    pmax=p(&x,&y);
    printf("max=%d\n",*pmax);
}
```

```
#include<stdio.h>
int main()
{
    int a,b,max(int,int),min(int,int),add(int,int);
    int (*p[3])()={max,min,add};
    printf("input a:");
    scanf("%d",&a);
    printf("input b:");
    scanf("%d",&b);
    printf("max=%d\n",p[0](a,b));
    printf("min=%d\n",p[1](a,b));
    printf("sum=%d\n",p[2](a,b));
    return 0;
}
max(int x,int y) {return x>y?x:y;}
min(int x,int y) {return x<y?x:y;}
add(int x,int y) {return x+y;}
```



动态分配内存

“动态内存分配”的概念

使用户程序能在运行期间动态的申请和释放内存空间，从而更有效地利用内存并提高程序设计的灵活性。

例如，为了保证程序的通用性，程序运行时需要的最长为10000个的元素的数组保存，但大部分运行的时候只需要30个左右的元素，于是大量分配的存储空间被浪费。

此时，可通过动态内存分配技术，将程序设计成运行时才向计算机申请内存，并在用完时立即释放占用的内存空间（堆和栈的概念）。



动态分配内存

以下函数在`malloc.h`或`stdlib.h`中定义（`n,x`为无符号整数，`p`为指针变量）：

❖ `void *malloc(x)`

分配一个长度为`x`字节的连续空间，分配成功返回起始地址指针，分配失败（内存不足）返回`NULL`

❖ `void *calloc(n,x)`

分配`n`个长度为`x`字节的连续空间（成败结果同上）

❖ `void *realloc(p,x)`

将`p`所指的已分配空间大小调整为`x`个字节

❖ `void free(p)`

将由以上各函数申请的以`p`为首地址的内存空间全部释放



动态分配内存

注意：

- ❖ 无论何时分配内存，必须要检查是否分配成功，即使申请的空间很小；
- ❖ 申请的内存使用结束后用`free()`函数释放，释放后应该把指向这块内存的指针指向NULL，防止程序后面不小心使用了它；
- ❖ 申请内存和释放内存这两个函数是配对函数，如果申请后不释放就是内存泄露。

```
int *pb=NULL;
pb=(int *)malloc(n*sizeof(int));
if(pb==NULL){
    puts( "memory allocation error." );
    exit(1);
}
```




动态分配内存

例 5.27 编程实现选择排序，假定事先不知道要排序的元素个数

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
void MakeArray(int v[],int n);
void Sort(int *v,int n);
void PrintArray(int v[],int n);
int main()
{
    int n,*pb=NULL;
    printf("input number of element:");
    scanf("%d",&n);
    pb=(int *)malloc(n*sizeof(int));
    if(pb==NULL)
    {
        printf("error!");exit(1);
    }
    MakeArray(pb,n);
    printf("before sort:\n");
    PrintArray(pb,n);
    Sort(pb,n);
    printf("after sort:\n");
    PrintArray(pb,n);
    free(pb);
    return(0);
}
```

```
void MakeArray(int v[],int n)
{
    int i;
    srand(time(NULL));
    for(i=0;i<n;i++)
        v[i]=rand()%1000;
}
void PrintArray(int v[],int n)
{
    int i;
    for(i=0;i<n;i++)
        printf("%5d",v[i]);
    printf("\n");
}
void Sort(int *v,int n)
{
    int t,i,j,k;
    for(i=0;i<n-1;i++)
    {
        k=i;
        for(j=i+1;j<n;j++)
            if(v[k]>v[j]) k=j;
        if(k!=i){t=v[i];v[i]=v[k];v[k]=t;}
    }
}
```



结构体与链表



特点：

- 按需分配内存
- 不连续存放
- 有一个“头指针”（head）变量
- 每个结点中应包括一个指针变量，用它存放下一结点的地址
- 最后一个结点的地址部分存放一个“NULL”（空地址）



结构体与链表



定义形式:

```
struct student
```

```
{
```

```
    int number;
```

```
    char name[6];
```

```
    struct student *next;
```

```
};
```

结构体指针成员



结构体与链表



链表常用操作

- `p=p->next` 在链表结点间顺序移动指针

将p原来所指结点中next的值赋给p，而p->next值即下一结点起始地址，故p=p->next 的作用是使p指向下一结点。

- `p2->next=p1` 将新结点添加到现在链表中

如果p2是链表中的末结点，p1指新建结点，此句的功能是使p1所指新结点变成链表中的新的末结点。

- `p2->next=NULL` 让p2所在结点成为链表中最后结点

例 5.28 若已建立下面的链表结构，指针p指向某单向链表的首结点

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next;
```

```
} *p=head;
```

以下语句能正确输出该链表所有结点的数据成员data的是_____。

☐ A for (;p!=NULL;p++) printf(“%7d,”,p->data);

☐ B for (;!p;p=p->next) printf(“%7d,”,(*p).data);

☒ C while(p) { printf(“%7d”,(*p).data); p=p->next; }

☐ D while (p!=NULL) { printf(“%7d”, p->data) ; p++; }

提交



结构体与链表

静态链表的建立

- ❖ 结点是如何定义的
- ❖ 结点是如何建立的
- ❖ 如何使诸结点形成链表
- ❖ 最后一个结点如何建立
- ❖ 如何从一个结点跳转到下一结点
- ❖ 如何遍历所有结点

```
#define NULL 0
struct student{
    long num;
    float score;
    struct student *next;
};
main() {
    struct student a,b,c,*head,*p;
    a.num=101;a.score=89.5;
    b.num=102;b.score=90;
    c.num=103;c.score=85;
    head=&a;
    a.next=&b;
    b.next=&c;
    c.next=NULL;
    p=head;
    while(p!=NULL){
        printf("%ld,%.1f\n",p->num,p->score);
        p=p->next;
    }
}
```



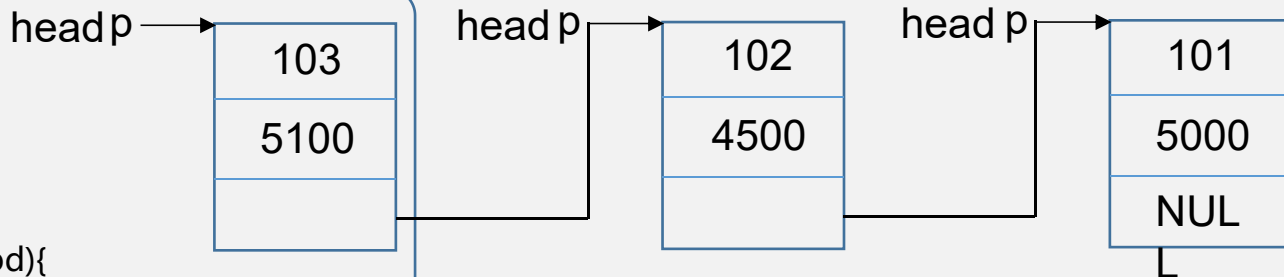
结构体与链表

动态链表的建立：头插法

no 103 head → NULL

```
struct person
{
    int num;
    float salary;
    struct person *next;
};

struct person *CreateList(viod){
    struct person *p=NULL;
    struct person *head=NULL;
    int no;
    printf("\n输入一个职工号,输入0结束:");
    scanf("%d",&no);
    while(no!=0){
        p=(struct person *)malloc(sizeof(struct person));
        p->num=no;
        printf("\n输入一个职工工资:");
        scanf("%f",&p->salary);
        p->next=head;
        head=p;
        printf("\n输入一个职工号, 输入0结束:");
        scanf("%d",&no);
    }
    return head;
}
```





结构体与链表

动态链表的建立：尾插法

```
struct person{
    int num;
    float salary;
    struct person *next;};
struct person *CreateList(viod){
    struct person *head=NULL;
    struct person *rear=NULL;
    struct person *p=NULL;
    int no;
    printf("\n输入一个职工号:");
    scanf("%d",&no);
    while(no!=0){
        p=(struct person *)malloc(sizeof(struct person));
        p->num=no;
        printf("\n输入一个职工工资:");
        scanf("%f",&p->salary);
        if(head==NULL) head=p;
        else rear->next=p;
        rear=p;
        printf("\n输入一个职工号，输入0结束:");
        scanf("%d",&no);
    }
    if(rear!=NULL) rear->next=NULL;
    printf("\n建表结束! \n");
    return head;
}
```




结构体与链表

遍历并输出动态链表

```
void PrintList(struct person *head){
    struct person *p=head;
    while(p!=NULL){
        printf("%d %f\n",p->num,p->salary);
        p=p->next;
    }
}
```

主函数，调用动态链表建立和输出函数

```
int main()
{
    struct person *head;
    head=CreateList();
    PrintList(head);
    return 0;
}
```



结构体与链表

动态链表的插入

```
struct person *ListInsert(struct person *head,struct person *ps)
//head为链表头指针，ps指向要插入的结点，由调用函数生成好
{
    struct person *p,*q;
    if(head==NULL) { //若为空表，使head直接指向插入结点皆可
        head=ps;
        printf("%d\n",head->num);
        return head;
    }
    if(head->num>ps->num){ //结点插入在表头
        ps->next=head;
        head=ps;
        return head;
    }
    p=q=head; //使p, q指向表头结点
    while(p!=NULL&& p->num<ps->num){ //寻找插入位置
        q=p;
        p=p->next; //使q指向p所指向的结点的前一个结点
    }
    q->next=ps; //插入新结点
    ps->next=p;
    return head;
}
```

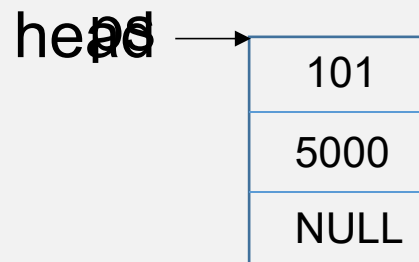


结构体与链表

动态链表的插入

情况一：初始链表为空表

head → NULL





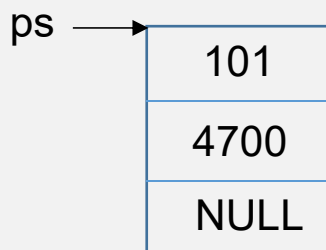
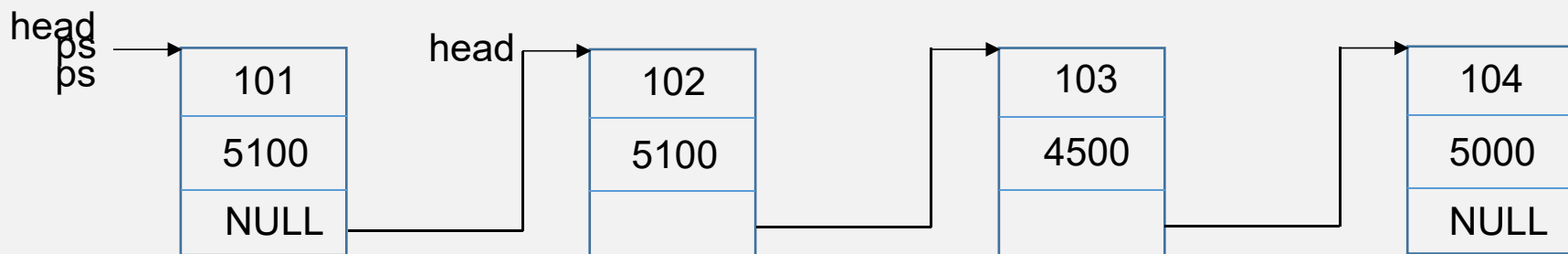
结构体与链表

动态链表的插入

情况二：节点插入在表头位置

① `ps->next=head;`

② `head=ps;`



条件判断： `ps->num < head->num`



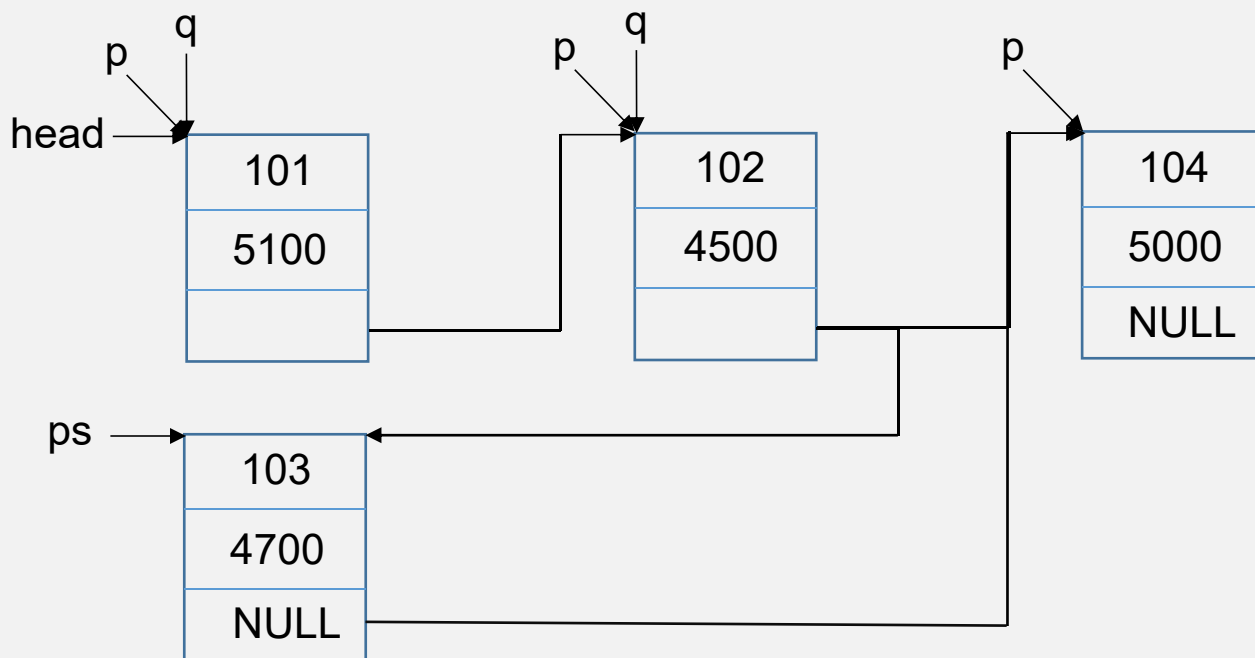
结构体与链表

动态链表的插入

情况三：节点插入不在表头位置



- ① $p = q = \text{head}$;
- ② 查找位置: $p \neq \text{NULL} \ \&\& \ p \rightarrow \text{num} < \text{ps} \rightarrow \text{num}$;
- ③ 移动指针: $q = p$; $p = p \rightarrow \text{next}$;
- ④ 节点插入: $q \rightarrow \text{next} = \text{ps}$; $\text{ps} \rightarrow \text{next} = p$;





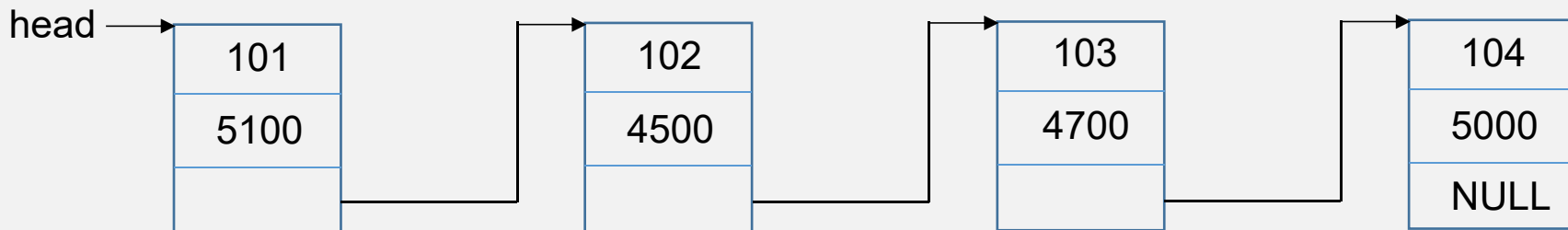
结构体与链表

动态链表的插入

情况三：节点插入不在表头位置



- ① $p=q=\text{head}$;
- ② 查找位置: $p \neq \text{NULL} \ \&\& \ p \rightarrow \text{num} < p_s \rightarrow \text{num}$;
- ③ 移动指针: $q=p$; $p=p \rightarrow \text{next}$;
- ④ 节点插入: $q \rightarrow \text{next}=p_s$; $p_s \rightarrow \text{next}=p$;





结构体与链表

动态链表的插入

情况三：节点插入不在表头位置



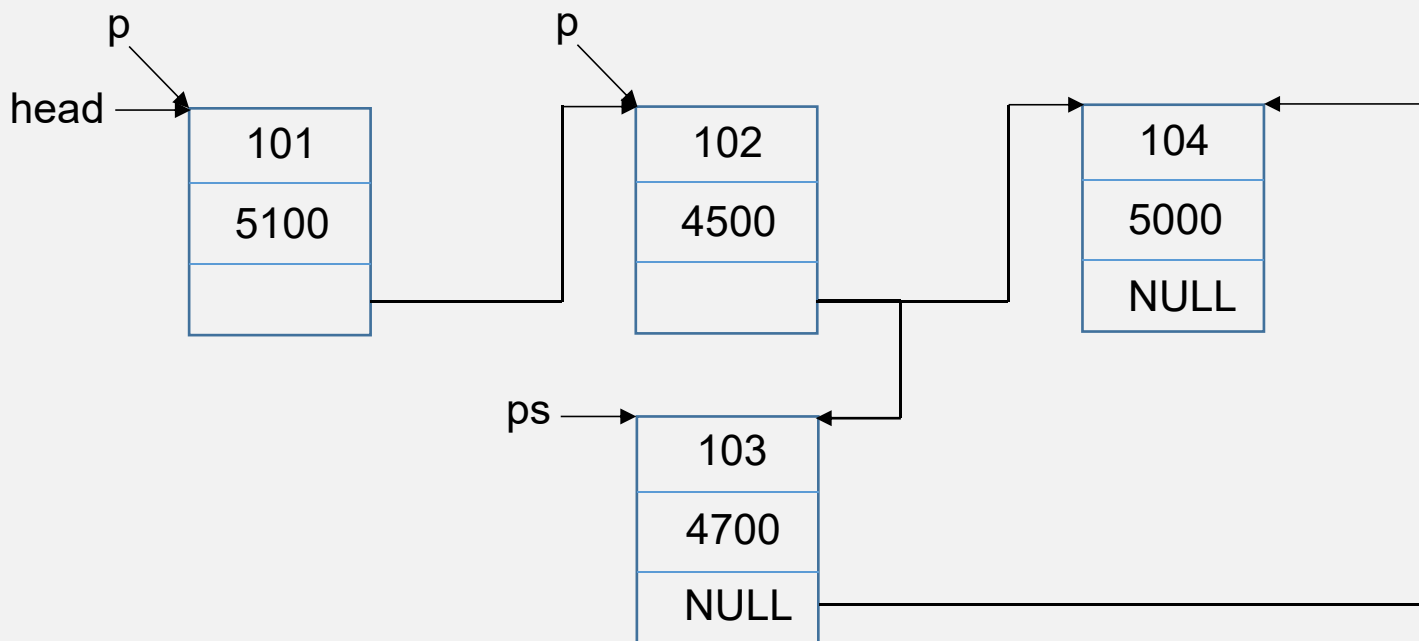
能否只用一个指针实现节点的插入？



结构体与链表

动态链表的插入

情况三：节点插入不在表头位置



- ① `p=head ;`
- ② 如何查找位置? `p->next!=NULL && p->next->num<ps->num ;`
- ③ 移动指针: `p=p->next ;`
- ④ 如何插入节点: `ps->next=p->next ; p->next=ps ;`



结构体与链表

动态链表的删除

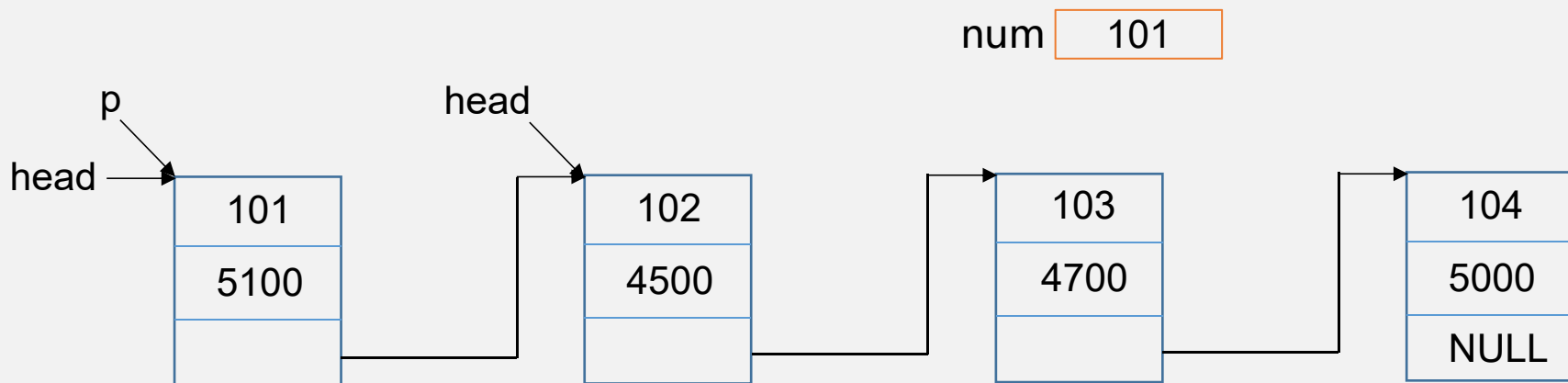
```
struct person *ListDelete(struct person *head,int num) {  
    struct person *q=NULL,*p=NULL;  
    if(head->num==num) {    //要删除的结点为链表首结点  
        p=head;  
        head=p->next;  
        free(p);  
        return head;  
    }  
    q=p=head;  
    while(p!=NULL&& p->num!=num) {    //查找要删除的结点  
        q=p;    //用q指向刚访问过的结点  
        p=p->next;    //使p指向下一个结点  
    }  
    if(p!=NULL) {    //查到要删除的结点  
        q->next=p->next;    //将p指向结点的下一个结点连接到p->next  
        free(p);    //删除p指向的结点  
        return head;  
    }  
    printf("is not found!\n");  
    return head;  
}
```



结构体与链表

动态链表的删除

情况一：要删除的节点是首节点



条件判断: $\text{head} \rightarrow \text{num}$ 等于 num

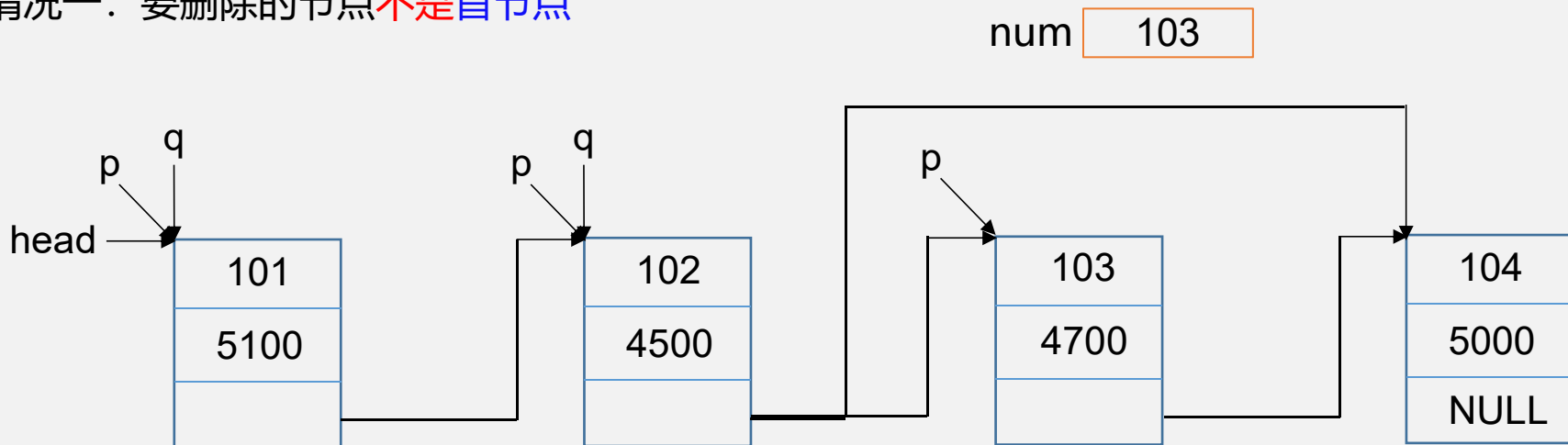
- ① $p = \text{head};$
- ② $\text{head} = p \rightarrow \text{next};$
- ③ $\text{free}(p);$



结构体与链表

动态链表的删除

情况一：要删除的节点不是首节点



- ① `p=q=head;`
- ② 查找位置: `p!=NULL && p->num!=num;`
- ③ 移动指针: `q=p; p=p->next;`
- ④ 节点插入: `q->next=p->next;`
- ⑤ `free(p);`



结构体与链表

动态链表的删除-头结点

```
struct person *ListDelete(struct person *head,int num) {  
    struct person *shead=NULL,*p=NULL,*temp=NULL;  
    shead=(struct person *)malloc(sizeof(struct person));    //创建头结点  
    shead->next=head;    //将头结点放在链表最前端  
    p=shead;  
    while(p->next!=NULL){    //头结点的存在避免了单独区分删除首个数据节点的情况  
        if(p->next->num==num){  
            temp=p->next;  
            p->next=p->next->next;  
            free(temp);  
        }  
        else{  
            p=p->next;  
        }  
    }  
    head=shead->next;  
    free(shead);    //释放虚拟头结点  
    return head;  
}
```



文件处理

❖ 打开文件fopen

- 函数原型： `FILE *fopen(char *name, char *mode)`
使用文件方式
- 功能：按指定方式打开文件
要打开的文件名
- 返回值：正常打开，为指向文件结构体的指针；打开失败，为NULL

```
例 FILE *fp;  
    fp= fopen ("test.dat","r");
```

文件使用方式	含义
"r/rb" (只读)	为输入打开一个文本/二进制文件
"w/wb" (只写)	为输出打开或建立一个文本/二进制文件
"a/ab" (追加)	向文本/二进制文件尾追加数据
"r+/rb+" (读写)	为读/写打开一个文本/二进制文件
"w+/wb+" (读写)	为读/写建立一个文本/二进制文件
"a+/ab+" (读写)	为读/写打开或建立一个文本/二进制文件



文件处理

```
FILE *fp;  
if((fp=fopen("test.c","w"))==NULL){  
    printf("File open error!\n");  
    exit(0);  
}
```

```
FILE *fp;  
fp=fopen("test.c","rb");  
if(fp==NULL){  
    printf("File open error!\n");  
    exit(0);  
}
```



文件处理

格式化读写：**fscanf** 与 **fprintf**

❖ 函数原型：

```
int fscanf(FILE *fp,const char *format[,address,...])
```

```
int fprintf(FILE *fp,const char *format[,argument,...])
```

功能：按格式对文件进行I/O操作

❖ 返回值：成功,返回I/O的个数;出错或文件尾,返回EOF



文件处理

例 5.29 从键盘按格式输入数据存到磁盘文件中

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

```
int main()
```

```
{
```

```
    char s[80],c[80];
```

```
    int a,b;
```

```
    FILE *fp;
```

```
    if((fp=fopen("test.txt","w"))==NULL)
```

```
    { printf("can't open file\n"); exit(0) ; }
```

```
    fscanf(stdin,"%s%d",s,&a);
```

```
    fprintf(fp,"%s %d",s,a);
```

```
    fclose(fp);
```

```
    if((fp=fopen("test.txt","r"))==NULL)
```

```
    { printf("can't open file\n"); exit(0); }
```

```
    fscanf(fp,"%s%d",c,&b);
```

```
    fprintf(stdout,"%s %d",c,b);
```

```
    fclose(fp);
```

```
    return 0;
```

```
}
```

打开文件

从键盘输入
向文件输出

关闭文件

再次打开文件

从文件输入
向显示器输出



文件处理

数据块（二进制形式）读写：**fread** 与 **fwrite**

❖ 函数原型：

```
size_t fread( void *buffer , size_t size , size_t count , FILE *fp )
```

```
size_t fwrite( void *buffer , size_t size , size_t count , FILE *fp )
```

❖ 功能：读/写数据块

❖ 返回值：成功，返回读/写的块数；出错或文件尾，返回0

❖ 说明：

- typedef unsigned size_t;
- buffer: 指向要输入/输出数据块的首地址的指针
- size: 每个要读/写的数据块的大小（字节数）
- count: 要读/写的数据块的个数
- fp: 已打开文件的地址
- fread与fwrite一般用于**二进制文件**的输入/输出



文件处理

```
float f[2];  
FILE *fp;  
fp=fopen("filename.dat","rb");  
if(fp==NULL){  
    printf("File open error!\n");  
    exit(0);  
}  
fread(f,sizeof(float),2,fp);
```

```
for(i=0;i<2;i++)  
    fread(&f[i],sizeof(float),1,fp);
```

```
struct student{  
    int num;  
    char name[20];  
    char sex;  
    int age;  
    float score[3];  
}stud[10]={ ... ... };  
for(i=0;i<10;i++)  
    fwrite(&stud[i],sizeof(struct student),1,fp);
```



文件处理

例 5.30 从键盘按格式输入数据存到磁盘文件中

```
#include <stdio.h>
#include <stdlib.h>
struct student{
    char name[10];
    int num;
    char addr[15];
}stud[3];
int main(){
    void save();
    void display();
    int i;
    for(i=0;i<3;i++){
        printf("第%d个学生姓名: ",i+1);
        scanf("%s",stud[i].name);
        printf("第%d个学生学号: ",i+1);
        scanf("%d",&stud[i].num);
        printf("第%d个学生籍贯: ",i+1);
        scanf("%s",stud[i].addr);
    }
    save();
    display();
    return 0;
}
```

```
void save(){
    FILE *fp;
    int i;
    if((fp=fopen("student_1.dat","w"))==NULL){
        printf("cannot open file\n");
        exit(0);
    }
    for(i=0;i<3;i++){
        if(fwrite(&stud[i],sizeof(struct student),1,fp)!=1)
            printf("file write error\n");
        fclose(fp);
    }
}

void display(){
    FILE *fp;
    int i;
    if((fp=fopen("student_1.dat","r"))==NULL){
        printf("cannot open file\n");
        exit(0);
    }
    for(i=0;i<3;i++){
        fread(&stud[i],sizeof(struct student),1,fp);
        printf("%s\t%d\t%s\n",stud[i].name,stud[i].num,stud[i].addr);
    }
    fclose(fp);
}
```



文件处理

例 5.30 从键盘按格式输入数据存到磁盘文件中



不使用全局数组



文件处理

例 5.30 从键盘按格式输入数据存到磁盘文件中

```
#include <stdio.h>
#include <stdlib.h>
struct student{
    char name[10];
    int num;
    char addr[15];
};
int main(){
    void save(struct student*);
    void display(struct student*);
    struct student stud[3];
    int i;
    struct student *p=stud;
    for(i=0;i<3;i++,p++){
        printf("第%d个学生姓名: ",i+1);
        scanf("%s",p->name);
        printf("第%d个学生学号: ",i+1);
        scanf("%d",&p->num);
        printf("第%d个学生籍贯: ",i+1);
        scanf("%s",p->addr);
    }
    p=stud;
    save(p);
    display(p);
    return 0;
}
```

```
void save(){
    FILE *fp;
    int i;
    if((fp=fopen("student_2.dat","w"))==NULL){
        printf("cannot open file,save failed\n");
        exit(0);
    }
    for(i=0;i<3;i++,q++){
        if(fwrite(q,sizeof(struct student),1,fp)!=1)
            printf("file write error\n");
        printf("保存成功! \n");
        fclose(fp);
    }
    void display(){
        FILE *fp;
        int i;
        if((fp=fopen("student_2.dat","r"))==NULL){
            printf("cannot open file,display failed\n");
            exit(0);
        }
        for(i=0;i<3;i++,q++){
            fread(q,sizeof(struct student),1,fp);
            printf("%s\t%d\t%s\n",q->name,q->num,q->addr);
        }
        fclose(fp);
    }
}
```



文件处理

例 5.31 存储、读取链表

```
void save(struct person *head){
    struct person *p=head;
    FILE *fp;
    char fname[50];
    printf("\n\n请输入要存储数据的文件名:");
    gets(fname);
    if((fp=fopen(fname,"wb"))==NULL){
        printf("文件无法打开, 存储失败!\n\n");
        exit(0);
    }
    while(p!=NULL){
        fwrite(p,sizeof(struct person),1,fp);
        p=p->next;
    }
    fclose(fp);
    printf("存储成功! \n\n");
}
```

存储



文件处理

例 5.31 存储、读取链表



读取



文件处理

feof

- 函数原型: `int feof(FILE *fp)`
- 功能: 判断文件是否结束
- 返回值: 文件结束, 返回真 (非0) ; 文件未结束, 返回0
- 判断二进制文件是否结束

```
while(!feof(fp))  
{  
    c=fgetc(fp);  
    .....  
}
```




文件处理

例 5.32 feof()函数

```
#include<stdio.h>
int main()
{
    FILE *fp;
    int i;
    if((fp=fopen("2.dat","r"))==NULL){
        printf("cannot open file\n");
        exit(0);
    }
    if (feof(fp))
        printf("文件为空");
    else
        printf("文件不为空");
    return 0;
}
```



文件处理

例 5.31 存储、读取链表

读取

```
struct person *load(){
    struct person *p,*s,*head=NULL;
    FILE *fp;
    char fname[50];
    printf("\n\n请输入要读取数据的文件名:");
    fflush(stdin);
    gets(fname);
    if((fp=fopen(fname,"rb"))!=NULL){
        printf("文件无法打开, 读取失败! \n\n");
        return;
    }
    while(!feof(fp)){
        if(head==NULL){
            p=(struct person *)malloc(sizeof(struct person));
            fread(p,sizeof(struct person),1,fp);
            if(feof(fp))//读取结束跳出循环
            {
                printf("文件为空! \n\n");
                return NULL;
            }
            head=s=p;
        }
        p=(struct person *)malloc(sizeof(struct person));
        fread(p,sizeof(struct person),1,fp);
        if(feof(fp))//读取结束跳出循环
        {
            s->next = NULL;
            break;
        }
        s->next=p;
        s=p;
        p->next=NULL;
    }
    fclose(fp);
    printf("读取成功! \n\n");
    return(head);
}
```



文件处理

字符读写 : fgetc 与 fputc

❖ 函数原型:

```
int fgetc( FILE *fp )
```

```
int fputc( int c , FILE *fp )
```

文件I/O与终端I/O

putchar(c)	fputc(c,stdout)
getchar()	fgetc(stdin)

- ❖ 功能: 从fp指向的文件中读取一字节代码/把一字节代码c写入fp指向的文件中
- ❖ 返回值: 成功, 返回读取/写入字符的ASCII码值, 出错返回-1 (EOF)



文件处理

例 5.33 从键盘输入字符，逐个存到磁盘文件中，直到输入'#'为止

```
#include <stdio.h>
int main()
{ FILE *fp;
  char ch;
  if((fp=fopen("out.txt","w"))==NULL)
  { printf("cannot open file\n");
    exit(0);
  }
  printf("input string:");
  while((ch=getchar())!='#')
  { fputc(ch,fp);
    putchar(ch);
  }
  fclose(fp);
  return 0;
}
```



文件处理

例 5.34 读文本文件内容，并显示

```
#include <stdio.h>
int main()
{ FILE *fp;
  char ch;
  if((fp=fopen("out.txt", "r"))==NULL)
  { printf("cannot open file\n");
    exit(0);
  }
  while((ch=fgetc(fp))!=EOF)
    putchar(ch);
  fclose(fp);
  return 0;
}
```



文件处理

例 5.35 实现在DOS环境下运行的文件拷贝命令 `copy 1.txt 2.txt`

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char **argv)
{
    void filecopy(FILE *fp1,FILE *fp2);
    FILE *fp1,*fp2;
    if(argc>1)
    {
        if((fp1=fopen(*++argv,"r"))==NULL){
            printf("can't open %s\n",*argv);
            exit(0);}
        if((fp2=fopen(*++argv,"w"))==NULL){
            printf("can't open %s\n",*argv);
            exit(0);}
    }
    filecopy(fp1,fp2);
    return 0;
}
void filecopy(FILE *fp1,FILE *fp2)
{ char c;
  while((c=fgetc(fp1))!=EOF)
    fputc(c,fp2);
}
```

复制文本文件



文件处理

例 5.35 实现在DOS环境下运行的文件拷贝命令 `copy 1.txt 2.txt`

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char **argv)
{
    void filecopy(FILE *fp1,FILE *fp2);
    FILE *fp1,*fp2;
    if(argc>1)
    {
        if((fp1=fopen(*++argv,"rb"))==NULL){
            printf("can't open %s\n",*argv);
            exit(0);}
        if((fp2=fopen(*++argv,"wb"))==NULL){
            printf("can't open %s\n",*argv);
            exit(0);}
    }
    filecopy(fp1,fp2);
    return 0;
}
void filecopy(FILE *fp1,FILE *fp2)
{
    char c;
    while(!feof(fp1)){
        c=fgetc(fp1);
        fputc(c,fp2);
    }
}
```

复制二进制文件



文件处理

字符串读写 : fgets 与 fputs

❖ 函数原型:

```
char *fgets(char *s,int n,FILE *fp)
```

```
int fputs(char *s,FILE *fp)
```

- ❖ 功能: **fgets**从fp所指文件读n-1个字符送入s指向的内存区,并在最后加一个'\0'
(若读入n-1个字符前遇换行符或文件尾(EOF)即结束)

fputs把s指向的字符串写入fp指向的文件

- ❖ 返回值: **fgets**正常返回读取字符串的首地址; 出错或文件尾, 返回NULL

fputs正常返回非负整数; 出错为EOF



文件处理

例 5.36 从键盘读入字符串存入文件，再从文件读回显示

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main()
{ FILE *fp;
  char string[81];
  if((fp=fopen("file.txt","w"))==NULL){
    printf("can't open file");exit(0); }
  while(strlen(gets(string))>0){
    fputs(string,fp);
    fputs("\n",fp);
  }
  fclose(fp);
  if((fp=fopen("file.txt","r"))==NULL){
    printf("can't open file");exit(0); }
  while(fgets(string,100,fp)!=NULL)
    fputs(string,stdout);
  fclose(fp);
  return 0;
}
```



文件处理

- ❖ 文件位置指针-----指向当前文件位置
- ❖ 读写方式
 - 顺序读写：位置指针按顺序移动
 - 随机读写：位置指针按需要移动

rewind函数

- ❖ 函数原型 `void rewind(FILE *fp);`
- ❖ 功能：重置文件位置指针到文件开头

例 对一个磁盘文件进行显示和写入

```
#include <stdio.h>

int main(){

    FILE *fp1,*fp2;

    fp1=fopen("test_1.c","r");

    fp2=fopen("test_2.c","w");

    while(!feof(fp1)) putchar(fgetc(fp1));

    rewind(fp1);

    while(!feof(fp1)) fputc(fgetc(fp1),fp2);

    fclose(fp1);

    fclose(fp2);

    return 0;

}
```



文件处理

fseek函数

- ❖ 函数原型 `int fseek(FILE *fp, long offset, int origin)`
- ❖ 功能：定位文件指针的位置
- ❖ **offset**：位移量（从起始点移动的字节数）正数向后移动，负数向前移动
- ❖ **origin**：起始点设定

文件开始	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件末尾	SEEK_END	2

ftell函数

- ❖ 函数原型 `long ftell(FILE *fp)`
- ❖ 功能：返回位置指针当前位置(用相对文件开头的位移量表示)
- ❖ 返回值：成功，返回当前位置指针位置；失败，返回-1



小结

- 在内存中直接操作数据与代码对象需要掌握系统级编程方法，**指针是基本的工具**
- 指针变量存储内存地址，可以用于间接访问内存中的数据和指令，特别是**指向数组、字符串、结构体、文件类型、动态分配存储空间、函数的指针**，以及指针用作函数参数和返回值的用法，使指针在系统级程序设计中起到重要作用
- 结构体是一类构造类型，用于描述由不同类型的属性组成的复杂对象，结构体和指针结合，**可以构建复杂的数据结构，如链表**，为提高存储与计算效率提供更多可能
- 文件操作使程序和磁盘文件建立起联系，能够**对磁盘文件进行读写操作**，扩展了程序中数据的来源和长期保存方法。

如同所有事物都有两个不同的方面一样，指针带来便利的同时，也对使用者提出了更高的要求。指针就像一把锋利的双刃剑，只有熟练地掌握它，才能充分地利用它所带来的各种好处，否则不仅可能使得程序难以理解，更可能会导致严重的运行错误。

能力要求

通过本章的学习，读者应达到如下能力水平：

- 针对学习与工作中遇到的系统级层面的问题，抽象出需要处理的数据对象，据此**构建合适的存储结构**；
- 在分析问题特征的基础上设计算法，灵活运用**指针、数组、结构体**等派生类型，结合**对磁盘文件的访问**，编写能解决更复杂的问题、实用价值更高的应用程序；
- 借助指针的系统级操作能力，对数据存储与算法运行效率进行最基础的优化。



中国科学技术大学

University of Science and Technology of China

提前预祝同学们寒假快乐！