

第 8 讲

间接访问：指针

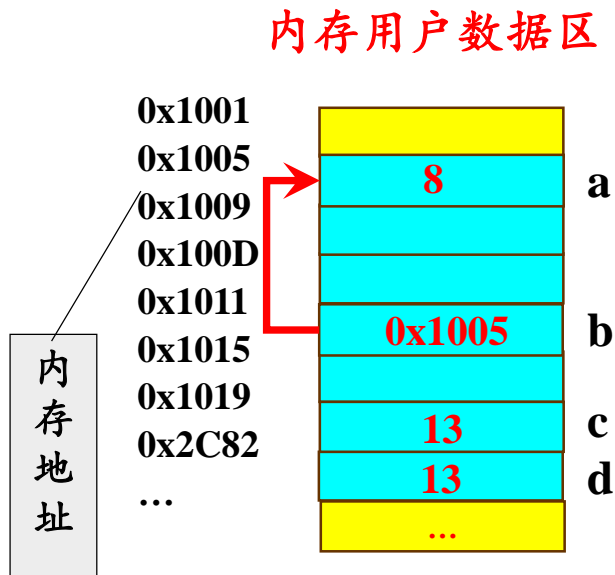
主要内容

- 8.1 指针的概念
- 8.2 指针变量的定义、初始化及引用
- 8.3 使用指针访问数组、指针数组
- 8.4 使用指针访问聚合数据
- 8.5 使用指针访问函数和文件

8.1 指针的概念

直接访问与间接访问

- 内存中每个字节都有一个编号，称为该字节的**地址**
- 程序运行时，每个变量都会被分配一块内存空间，其起始字节的编号，称为该**变量的地址**
- **直接访问**：通过变量名（对应一个地址）访问内存空间中的数据（**用钥匙打开抽屉直接取物品**）
- **间接访问**：通过第二个变量访问第一个变量所对应的内存空间中的数据（**抽屉b中存放着抽屉a的钥匙，通过b抽屉去取a抽屉中的物品**）



```
int a, c, d;
```

直接访问:

```
a=8; c=a+5;
```

```
printf("c=%d", c);
```

c=13

间接访问:

```
int *b;
```

```
b=&a; d=*b+5;
```

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

d=13

- 通过内存单元的地址能够在内存中找到该内存单元，其作用就像公园里指针形状的路标。因此，将地址形象化地称为“**指针**”（**pointer**），它指向地址所标识的内存单元（变量的指针就是变量的地址）
- 必须定义一个变量专门存放指针，该变量称为**指针变量**

8.2 指针变量的定义、初始化及引用

指针变量的定义

格式：

基类型名 *指针变量名

//基类型——该指针变量所指向的地址中存储的数据

//的类型

例如： **int *pi1, *pi2, *pi3;**

float *pf1, *pf2, *pf3;

char *pc1, *pc2, *pc3;

指针变量的初始化

格式: **基类型名 *指针变量名=&指向的变量名;**

例如: **int i; int *pi=&i;**

float f; float *pf=&f;

char c; char *pc=&c;

指针变量的引用

格式:

***指针变量名**

例8-1: `#include<stdio.h>`

```
void main( )  
{  int i, *pi=&i;  
    scanf("%d", pi);  
    printf(" i=%d,*pi=%d\n", i, *pi);  
    *pi=314;  
    printf(" i=%d,*pi=%d\n", i, *pi);  
}
```

```
123  
i=123, *pi=123  
i=314, *pi=314
```


注：

- (1) 定义指针变量后，如果不赋初值，该指针变量无具体的指向，即指向不确定（**悬空指针**）。通过悬空指针访问内存区，可能会导致难以预料后果。**解决办法**：将其赋值为**NULL**（0，它可以赋给任何类型的指针变量）
- (2) 可以定义void类型指针变量。void类型指针是一种特殊的指针，它无须类型转换即可指向任意类型指针；任意类型的指针也可指向void类型

```
#include<stdio.h> // 例8-2
```

```
void main( )
```

```
{
```

```
    double f=1.23, *pf;
```

```
    void *p;
```

```
    p=&f;
```

```
    pf=(double *)p;
```

```
    printf("&f=%x, p=%x, pf=%p\n", &f, p, pf);
```

```
    printf("*pf=%f\n", *pf);
```

```
}
```

```
&f=19ff38, p=19ff38, pf=0019FF38
*pf=1.230000
```

指针的基本操作

- **求地址（&）**：求被操作对象的地址（数组的起始地址用**数组名**获取）
- **取内容（*）**：访问地址表达式所指向的变量的值
- **赋值（=）**：将一个指针值赋给某个指针变量

char c; char *pc=&c, *pc1=pc;

注：赋值运算要求赋值号右侧的指针值与赋值号左侧的指针变量之间类型相容，不同类型的指针变量之间不能赋值

■ 算术运算：

- ✓ 指针与整数进行加减运算：不是简单的加减法
- ✓ ++/--运算：不是简单的自加/自减
- ✓ 指针变量减法运算：通常用于指向同一数组元素

■ 关系运算：指向同一数组时才有意义

■ 类型转换：

格式

(类型名 *) 指针变量名

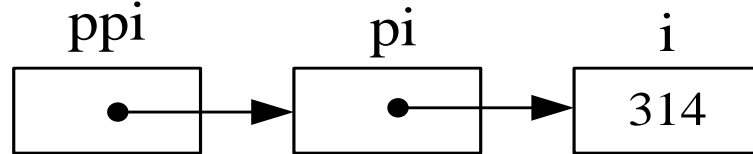
多级指针

指针变量的值也需要存放在内存的某个区域，它也有类型、起始地址。可以另外定义一个指针变量来存放其起始地址，这个指针称为指向指针的指针（多级指针）

定义格式：

```
基类型名 **指针变量名[=&指针变量名];
```

例如: `int i=314, *pi, **ppi;`
`pi=&i;`
`ppi=π`



要访问变量*i*的内容，有三种方式：

- (1) 直接访问*i*
- (2) 通过指针变量*pi*间接访问*i*: `*pi==i`
- (3) 通过指针变量*ppi*间接访问*i*: `**ppi==i`

指针变量作为函数的参数

■ 指针变量作为函数的形参

- ✓ 将一个实参变量的地址传递到一个函数中
- ✓ 形参定义为与实参同类型的指针变量
- ✓ 实现对普通变量的引用传递

指针传递可以降低参数传递的代价，使主调函数与被调函数共享同一块内存空间

```
#include<stdio.h>           // 例8-3
void fun(int a, int b, int *p, int *q)
{
    *p=a+b;
    *q=a-b;
}
void main( )
{
    int a=111, b=333, sum, sub;
    fun(a, b, &sum, &sub);
    printf("sum=%d, sub=%d\n", sum, sub);
}
```

sum=444, sub=-222

■ 指向指针的指针变量作为函数的形参

- ✓ 实现对指针类型变量的引用传递
- ✓ 函数调用时，将实参指针变量本身的地
址传递给形参
- ✓ 形参定义为二级指针变量

`#include<stdio.h>` **//例8-4程序功能：交换两个变量的值**

`void exch(int **p, int **q)`

```
{   int t;  
    t=**p;  
    **p=**q;  
    **q=t;  
}
```

`void main()`

```
{  
    int a=28, b=75;  
    printf("a=%d, b=%d\n", a, b);  
    int *pi1=&a, *pi2=&b;  
    exch(&pi1, &pi2);  
    printf("a=%d, b=%d\n", a, b);  
}
```

a=28,	b=75
a=75,	b=28

8.3 使用指针访问数组

使用指针访问一维数组

如： `int *p, a[10];`

`p=a;` //使指针变量p指向数组

	下标法	地址法	指针法
第k个元素的地址	<code>&a[k]</code>	<code>a+k</code>	<code>p+k</code>
第k个元素	<code>a[k]</code>	<code>*(a+k)</code>	<code>*(p+k)</code>

注：

- (1) 要通过指针处理数组，首先应该使指针指向该数组，具体方式为：`p=a;` 或 `p=&a[0];`
- (2) 指针变量是变量，值可以改变，数组名是常量，值不能改变。如：`p++;` 正确 `a++;` 错误
- (3) 指向同一数组的两个指针变量相减，为这两个指针变量所指向的数组元素之间跳过的元素个数。两个指针之间不能进行加、乘及除法运算
- (4) 指针变量也可以带下标，如：`*(p+k)` 等价于 `p[k]`

例8-5:

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

```
void main()
```

```
{
```

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

```
    printf("下标法输出各元素: \n");
```

```
    for(i=0; i<10; i++)    printf("%4d", a[i]);
```

```
    printf("\n指针法输出各元素: \n");
```

```
    for(p=a; p<a+10; p++) printf("%4d", *p);
```

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

```
}
```

下标法输出各元素:

1 2 3 4 5 6 7 8 9 10

指针法输出各元素:

1 2 3 4 5 6 7 8 9 10

也可以通过指针数组或指向指针的指针变量处理数组

□ 指针数组：数组元素全部为指针类型，每个元素都相当于一个指针变量

□ 格式： **基类型名 *数组名[长度]={地址表列};**

例8-6:

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

```
void main()
```

```
{
```

```
    int i, a[5];
```

```
    int *p[5]={ &a[0],&a[1],&a[2],&a[3],&a[4]};
```

```
    for(i=0; i<5; i++) scanf("%d", p[i]);
```

```
    for(i=0; i<5; i++) printf("%3d", *(p[i]));
```

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

```
}
```

```
10 20 30 40 50
10 20 30 40 50
```

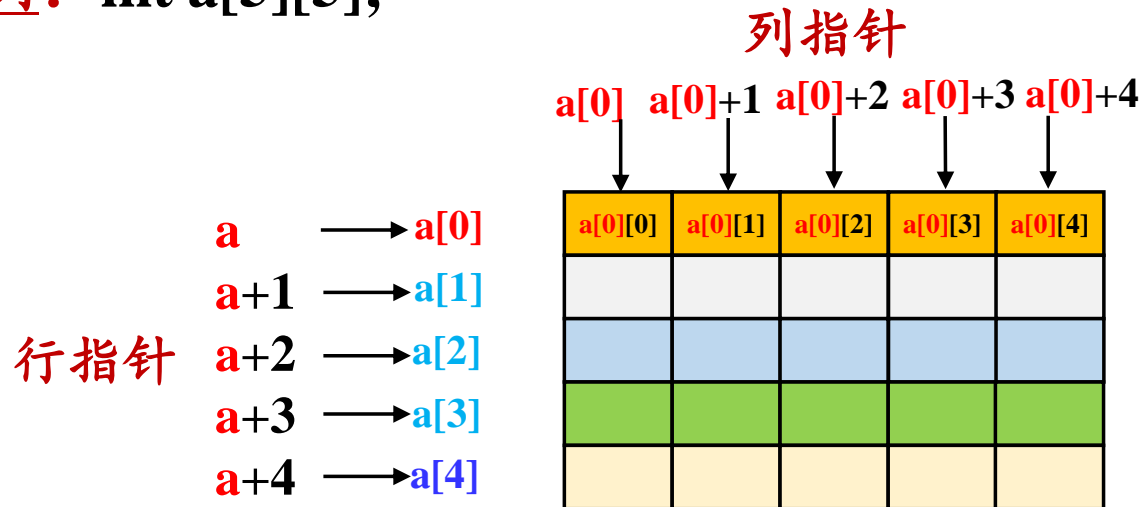
例8-7:

```
#include<stdio.h>
void main()
{
    int i, a[5], *p[5]={ &a[0],&a[1],&a[2],&a[3],&a[4]};
    int **pt=p;
    for(i=0; i<5; i++) scanf("%d", *pt++);
    pt=p;
    for(i=0; i<5; i++) printf("%3d", *(*pt++));
    printf("\n");
}
```

```
10 20 30 40 50
10 20 30 40 50
```


使用指针访问二维数组

例: `int a[5][5];`



如：int a[3][4], *p;

p=a[i]; // **p**每次跳动一个元素，为列指针变量

	下标法	地址法	列指针法
第i行、j列元素的地址	&a[i][j]	a[i]+j或*(a+i)+j	p+j
第i行、j列元素	a[i][j]	*(a[i]+j)或*(*(a+i)+j)	*(p+j)

行指针变量（指向一维数组的指针变量）

定义格式：**基类型名 (*指针变量名)[长度];**

如：int (*p)[4]; // p的基类型为int [4]，即具有4个
// int型元素的一维数组

int a[3][4]; **p=a;**

	下标法	地址法	行指针法
第i行、j列元素的地址	&a[i][j]	a[i]+j或*(a+i)+j	*(p+i)+j
第i行、j列元素	a[i][j]	*(a[i]+j)或*(*(a+i)+j)	*(*(p+i)+j)

例8-8:

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

```
void main( )
```

```
{
```

```
    int i, j, a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12}, (*p)[4];
```

```
    p=a;
```

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

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

```
    { for(j=0; j<4; j++)    printf("%4d", *((p+i)+j));
```

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

```
    }
```

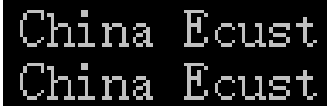
```
}
```

1	2	3	4
5	6	7	8
9	10	11	12

使用指针处理字符串常量

例8-9:

```
#include<stdio.h>
void main( )
{
    char *p;
    p=" China Ecust";
    puts(p);
    for(; *p!='\0'; p++)
        printf("%c", *p);
}
```

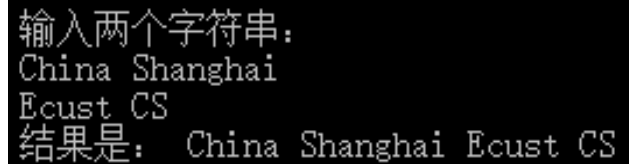


```
China Ecust
China Ecust
```

使用指针访问字符数组

例8-10:

```
#include<stdio.h>
void main( )
{
    char str1[50], str2[20], *p, *q;
    printf("\n 输入两个字符串: \n");
    gets(str1); gets(str2); p=str1; q=str2;
    while(*p) p++;
    while(*p++=*q++);
    printf(" 结果是: %s\n", str1);
}
```



```
输入两个字符串:
China Shanghai
Ecust CS
结果是: China Shanghai Ecust CS
```

使用字符指针变量和字符数组的比较

	字符数组	字符指针变量
存储方式不同	分配一段连续的空间	分配一个用于存放地址的存储区域
运算方式不同	数组名是指针常量，不能运算	字符指针变量是指针变量，可以运算
赋值方式不同	可以初始化，不能用赋值语句整体赋值	可以初始化，可以用赋值

8.4 使用指针访问结构体

结构体指针变量的定义与初始化

定义格式：

结构体类型 *指针变量名；

如： **struct Student**

```
{ int num; char name[20]; float score; };
```

```
struct Student st1={1001, “张三”, 85};
```

```
struct Student *p=&st1;
```

或

```
struct Student *p; p=&st1;
```


结构体指针变量的引用

与基本数据类型指针变量的引用方式类似

引用形式一：(*结构体指针变量).成员名

引用形式二：结构体指针变量->成员名

注：

- (1) 由于成员运算符“.”的优先级比指针运算符“*”高，*p两侧应加“()”，即(*p).num
- (2) “->”称为指向运算符，该运算符只能用于结构体指针变量，直接获得成员变量。即p->num与(*p).num等价
- (3) C允许一个结构体类型的成员中含有指向同一结构体类型的指针，这种结构称为自引用结构
如： `struct abc{ int a; float b; struct abc *p;}`

如：

(*p).num++;

(*p++).num; //引用**p**指向的结构体变量中成员**num**的值，再使**p**增1

(*++p).num; // ?

p->num++; //引用**p**指向的结构体变量中成员**num**的值，再使**num**增1

(p++)->num; //引用**p**指向的结构体变量中成员**num**的值，再使**p**增1

++p->num; //使**p**指向的结构体变量中成员**num**的值先增1，再引
//用**num**的值，不是使**p**增1

使用指针访问结构体数组

如: **struct Student**

```
{ int num; char name[20]; float score; };
```

```
struct Student st[10];
```

```
struct Student *p=st;
```

引用时: 可以 st[0].name 、 (*p).name 、 p->name

p++; 表示p指向st[1]元素

8.5 使用指针访问函数及文件

返回指针值的函数

定义格式: **类型名 *函数名(参数表列);**

如: `int *fun_p(int a, float b, char c);`

或 `int *fun_p(int, float, char);`

这里, 定义了一个函数, 该函数具有三个不同类型的参数, 函数的返回值是一个指针值

函数指针变量的定义与初始化

定义格式: **类型名 (*指针变量名)(参数表列);**

如: `int (*p)(int a, float b, char c);`

或 `int (*p)(int, float, char);`

这里, 定义p为一个指向具有三个不同类型参数、返回值为int的函数的指针变量

例8-11:

```
#include<stdio.h> //斐波那契数列
```

```
int fun(int n)
```

```
{ if(n==1||n==2) return 1;;
```

```
    return fun(n-1)+fun(n-2);
```

```
}
```

```
void main( )
```

```
{
```

```
    int n; long m; int (*p)(int); p=fun;
```

```
    printf("请输入要求的是第几项: ");
```

```
    scanf("%d", &n);
```

```
    m=(*p)(n);
```

```
    printf("%d\n", m);
```

```
}
```

```
请输入要求的是第几项: 20
6765
```

指针作为main()函数的形参

- 通常main函数和其他函数组成一个文件模块，有一个文件名
- 对这个文件进行编译和连接，得到可执行文件。用户执行这个可执行文件，操作系统就调用main函数，然后由main函数调用其他函数，从而完成程序的功能

- `main`函数的形参是从哪里传递给它们的呢？
- 由于`main`函数是操作系统调用的，实参只能由操作系统给出

一般形式：**`main (int argc, char *argv[])`**

其中：`argc`是命令行实参的个数，字符指针数组
`argv`接收命令行上各字符串的首地址

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

```
China
Shanghai
Ecust_CS
```

方法一：在VC++环境下编译、连接后，“工程”——“设置”——“调试”——“程序变量”中输入“China Shanghai Ecust_CS”，再运行就可得到结果

方法二：DOS虚拟机下运行

Homework

- 实践

实验6 指针程序设计

- 作业

《教材》： P291-292 3、 9、 11、 20