算法语言程序设计

—C语言程序

指针

主要内容：

指针的概念

指针运算

指针与函数参数

指针与数组

字符指针和字符串

函数型指针

指针型指针

指针功能

指针是一种构造类型数据，利用指针可以有效地表示复杂的数据结构，动态地分配内存，方便地使用字符串和数组，灵活地实现函数间的数据传递，直接处理内存地址等。

掌握指针的应用，可以使算法表达更确切，程序更清晰简炼，代码 更紧凑有效。

7.1 指针的概念

7.1.1 存储单元的地址与内容

程序一旦被执行，则该程序的指令、常量和变量等都要存放在机器的内存中。

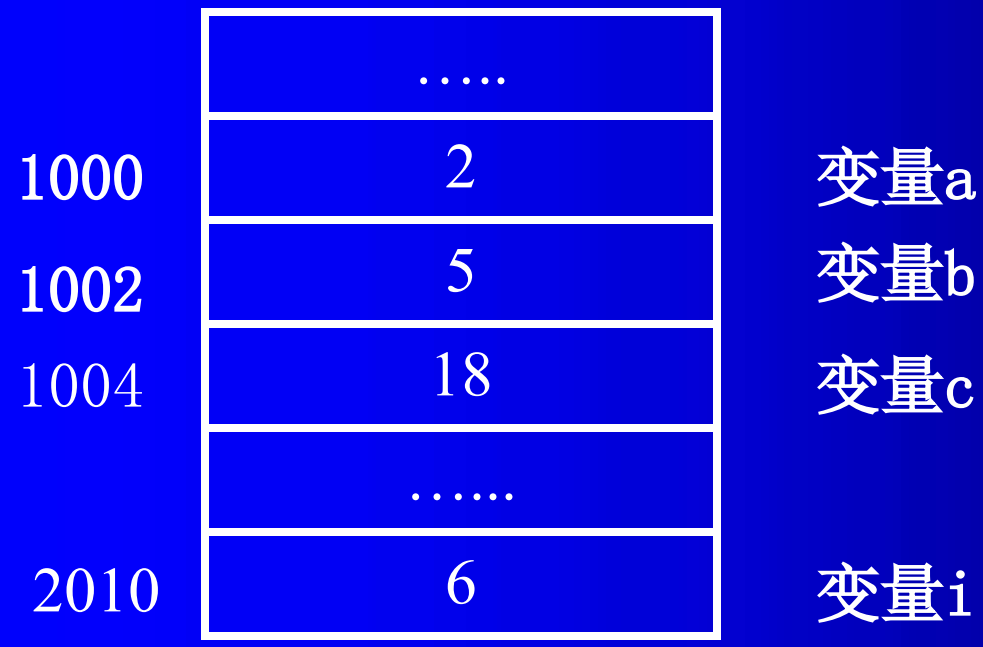
内存是以字节来划分存储单元的，每个字节都有一个编号，这就是存储单元的“地址”。

在计算机中 ，根据内存的地址，就可以访问存储在该单元中的数据。

将某存储单元中的数据称为这个存储单元的“内容”。

存储单元的地址与内容如下图所示。

图7.1　内存



7.1.2 指针和指针变量

内存变量的三个基本特征：

变量名、变量的内容和变 量的存储地址。

要访问一个变量，可以有两种方式：

用变量名；用变量的地 址。

7.1.3 指针变量的定义

指针变量使用前应先定义。

定义格式：

类型标识符　*标识符

说明：

① 标识符是被定义的指针变量的名字。

② 类型标识符则表明了该指针变量所指向变量的类型。

例如：int *p;　　char *ch;

7.1.4 指针运算符和指针变量的引用

1. 运算符

& 取地址运算符

运算格式：&标识符

* 指针运算符(间接访问运算符)

运算格式：*指针变量

例如：

```
int *p, i, j;
…
p=&i;
j=*p;
```

表示把变量i的地址赋给指针变量

表示将指针变量所指向的变量i的内容赋给变量j

2. 指针变量的引用

指针变量一经定义，可以像其它基本类型变量一样引用。

(1) 指针变量只接收地址，例如：

```
int *pi,*pj,*pk,i,j,k;
...
pi=&i;
pj=&j;
pk=&k;
```

(2) 指针变量指向变量后，可以像其它基本类型变量一样引用。

例如：

```
int *pi,*pj,i,j;
pi=&i;
pj=&j;
*pi=0;   /*等价于i=0;*/
*pj+=1; /*等价于j+=1; */
*(pi)++; /*等价于i++; */
```

切记：指针变量引用前必须指向某个变量。

例如：

```
#include〈stdio.h〉
void main()
{ int *p;
  *p=100;        /* 错*/
  printf("%d \n",*p);
}
```

(3)指向同类型的指针变量之间可以相互赋值

例如：pi=pj;

注意：对于指针变量pi和pj，下面两种赋值的效果是不等价的。

pi=pj;　　和　　*pi=*pj;

(4) 如果指针变量pi已经指向变量i，那么：

&*pi与&i等价。

因为两个运算符的优先级别相同，按自右向左方向结合，先进行*pi运算，它就是变量i，再进行&运算，所以&*pi与&i等价。

(5) 若i是一变量，则*&i与i等价。

因为先进行&i运算，其结果是地址，再进行*运算，相当于取这个地址中的内容，即i的值。故*&I与i等价。

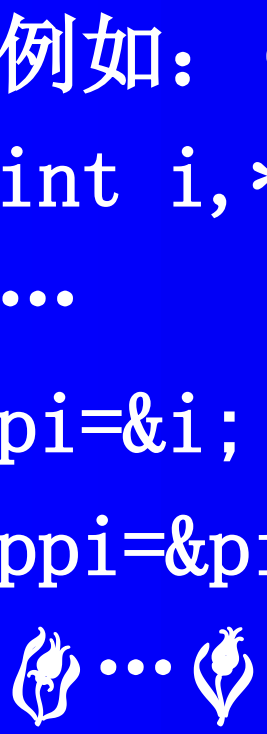
(6) 指针变可以用存储类型说明，对于被说明为静态的和外部的指针变量可以被赋初值(初始化)。

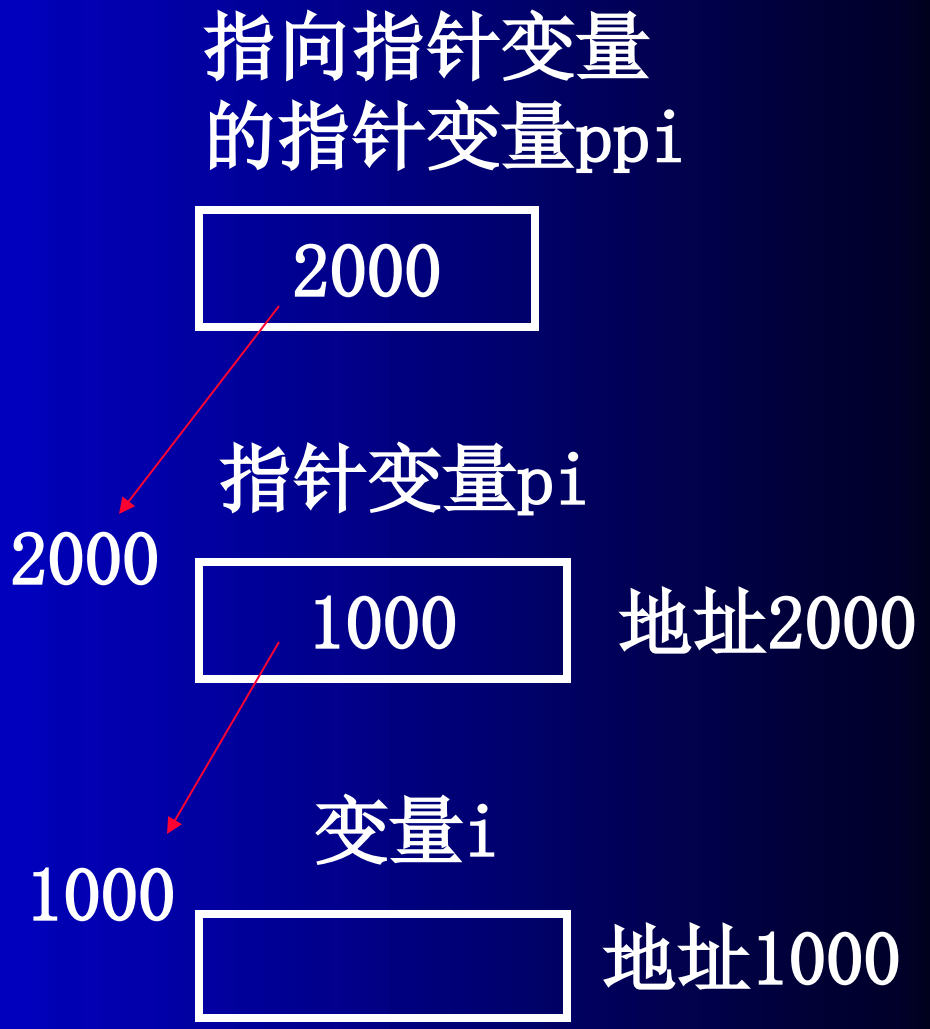
例如：
```
int *p,i,*pi=&i;
#include 〈stdio.h〉
void main()
{ static char a[10],*pa=a;
  …    }
```

(7) 指针变量可以指向任何类型变量，其中包括其他的指针变量。

例如：

```
int i,*pi,**ppi;
…
pi=&i;
ppi=&pi;
```





(8)指针变量可以赋“空”值，其含义是该指针变量不指向任何变量。

“空”值通常作为指针的异常标志。用NULL表示，即全部二进位均为0值。NULL其实就是0，但习惯上不用0而用NULL，办法是通过#define定义。

例如：

```
#define NULL 0
#include 〈stdio.h〉
void main()
{  int i,*p;p=NULL;  …
   p=i;…
   if(p==NULL) printf(" pointer p is NULL");
   … }
```

(9) 一个指针变量的值为NULL与未对该指针变量赋值是不同的。

前者是有值的，其值为0，称“空值”，它表示不指向任何变量；而后者虽未对指针变量赋值，但并不等于该指针变量无值，只是它的值是一个不确定的值，即该指针变量正指向某个未知的单元。这时，程序若引 用这个指针变量，显然是很危险的。所以，在引用一个指针变量前，必须先要对它赋值。

3 指针变量引用举例

【例7.1】观察指针变量pi和pj构成的语句(pi=pj;*pi=*pj;)之异同。

```
#include 〈stdio.h〉
void main()
{int i,j,*pi,*pj;
 scanf("%d,%d",&i,&j);
 pi=&i;pj=&j;
 printf("i=%d, j=%d\n",i,j);
 printf("*pi=%d,*pj=%d\n",*pi,*pj);
 printf("pi=%u,pj=%u\n",pi,pj);
 pi=pj;
 printf("i=%d, j=%d\n",i,j);
 printf("*pi=%d,*pj=%d\n",*pi,*pj);
 printf("pi=%u,pj=%u\n",pi,pj);
 pi=&i;pj=&j;*pi=*pj;
 printf("i=%d, j=%d\n",i,j);
 printf("*pi=%d,*pj=%d\n",*pi,*pj);
 printf("pi=%u,pj=%u\n",pi,pj);}
```

程序运行结果如下：

```
2,8
i=2, j=8
*pi=2,*pj=8
pi=65494,pj=65496
i=2, j=8
*pi=8,*pj=8
pi=65496,pj=65496
i=8, j=8
*pi=8,*pj=8
pi=65494,pj=65496
```

【例7.2】 输入a、b两个整数，经比较后，按大小顺序输出a和b。

```
#include〈stdio.h〉
void main()
{ int *pi,*p2,*p,a,b;
  scanf("%d,%d",&a,&b);
  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)
  ;
```

程序运行结果如下：

```
9,18
a=9,b=18
max=18,min=9
```

7.1.5 指针数组

指针数组：定义为指针型的数组称为指针数组。

例：利用指针数组对字符型二维数组输入输出。

```
#include〈stdio.h〉
void main()
{  char a[5][20],*p[5];
   int i;
   for(i=0;i<5;i++)
      p[i]=a[i];
   for(i=0;i<5;i++)
      scanf(“%s”,p[i]);
   for(i=0;i<5;i++)
      printf(“%s\n”,p[i]);
 }
```

7.1.6 数组行指针

数组行指针：用于指向二位数组的某一行的指针称为数组行指针。

格式： int (*p)[m];

说明：① p是行指针变量； ② m是数组列大小。

- 数组元素地址 *(指针变量+行下标)+列下标
- 数组元素引用 *(*(指针变量+行下标)+列下标)

例：

```
#include〈stdio.h〉
   void main()
   {  int a[2][3],(*p)[3]=a; int i,j;
      for(i=0;i<2;i++)
      for(j=0;j<3;j++)   scanf(“%d”,*(p+i)+j);
      for(i=0;i<2;i++)
      {  printf(“\n”);
         for(j=0;j<3;j++)
            printf(“%10d”,*(*(p+i)+j));
      }  }
```

7.2 指针运算

指针:是指向某类型数据的地址。

说明:

- 地址的分配是由C编译系统决定的;
- 指针变量的值是一具体类型的特定变量地址所允许的整数，但必须明确：指针不是整数。
- 对指针的运算不能像对整型数据那样进行所有的算术、逻辑和关系运算，而只能进行C语言所规定的某些运算。
- 指针运算的值与某一类型数据的地址有关。

7.2.1 指针的算术运算

设p、q是指针变量，n为一整数，则：p+n、p-n、p++、p--、++p、--p、p-q都是指针变量允许实施的算术运算，它们的意义分别是：

(1) p+n：表示由p所指向位置向高地址移n个位移量。

(2) p-n：表示由p所指向位置向低地址移n个位移量。

(3) p++：将当前指针p向高地址移一个位移量

(4) p--：将当前指针p向低地址移一个位移量。

(5) ++p：将当前指针p向高地址移一个位移量。

(6) --p：将当前指针p向低地址移一个位移量。

(7) p-q：表示两个被指向对象间相隔位移量的个数。

【例7.3】 计算字符串长度函数。

```
strlen(char *s)
{   char *p=s;

  while(*p!='\0')
       p++;
    return(p-s);
}
```

```
strlen(char *s)
{   char *p=s;
    while(*p)
       p++;
    return(p-s);
}
```

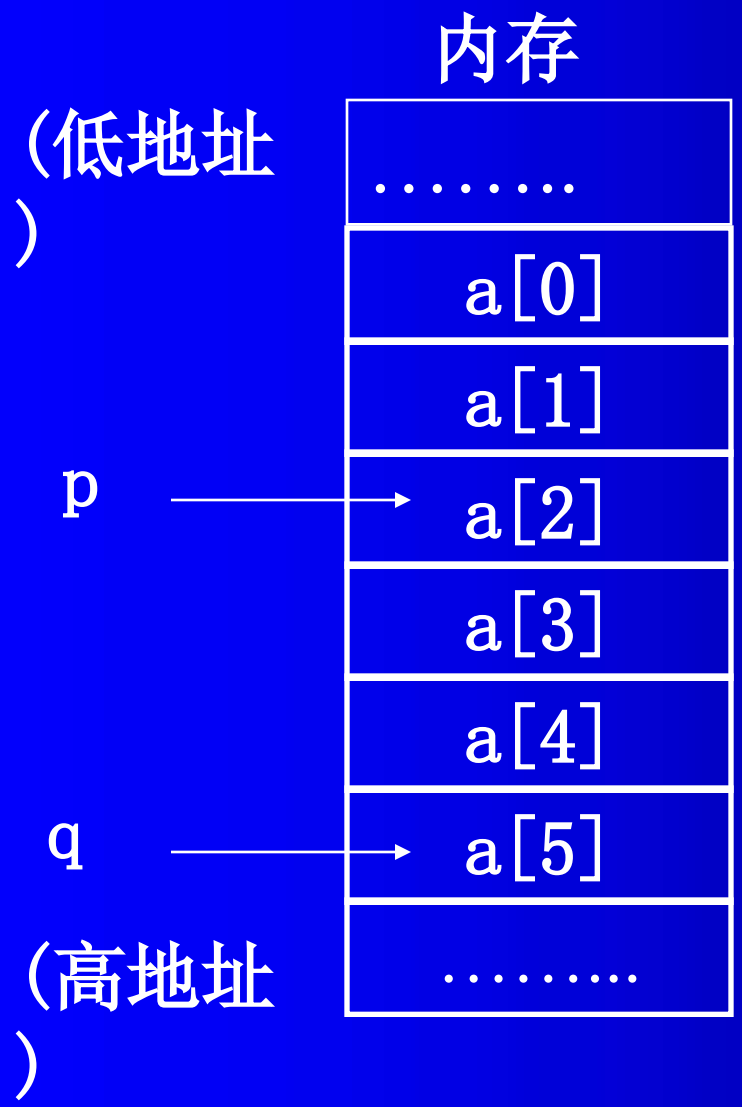
7.2.2 指针的关系运算

设p、q是指向同一数据集合的指针变量，则p<q、p<=q、p>q、p>=q、p==q、p!=q是指针变量允许实施的关系运算。它们的意义分别是：

（1）p>q：若表达式的结果为非0；则表明p指针变量所指向的元素在q指针变量所指向的元素之后。否则，结果为0，则表明p指向的元素在q指向的元素之前。

（2）p<q：若表达的结果为非0，则表明p所指向的元素在q所指向的元素之前。否则，结果为 0，则表明p所指向的元素在q所指向的元素之后。

图7.7



7.2.3 指针变量的赋值运算

设p,q是指向同一数据类型的指针变量，n为一整数，则：p=q、p=q+n、p=q-n、p+=n,p-=n都是指针变量允许实施的赋值运算。它们的意义分别是：

(1) p=q：将q中的地址值赋给p。

(2) p=q+n：将由q所指位置向高地址移n个位移量后，所得的实际地址值赋给p。

(3) p=q-n：将由q所指位置向低地址移n个位移量后，所得的实际地址值赋给p。

(4) p+=n：等价于p=p+n。

(5) p-=n：等价于p=p-n。

指针变量在进行赋值运算时，要注意：

(1) 相互赋值的指针变量p、q，它们所指向的变量的类型应一致，否则会出错。

(2) 如果p+=x中，x低于int类型级别时，应将x强制转换成int类型。如x为double型时，写为：p+=(int)x;

(3) 指针的赋值运算仅上述列出的形式有意义，其他均无意义。更不能将一个整型变量的值或常数赋给指针变量。即：

p=n;　　p=100;都是错误的。

7.2.4 指针运算符与自增、自减运算符的混用

指针运算符和自增、自减运算符都属于同优先级左结合的单目运算符。它们的混用在程序设 计中较为普遍，尤其是指针运算符*与自增运算符++、

自减运算符--的混用更为常见。例如 ：

```
while(*p)  putchar(*p++);
```

中的*p++就是指针运算符与自增运算符混用的一种。在这条语句中，*p++的运算结果是：先将*p作为操作数输出，再将指针变量p向高地址自增一个位移量。所以，*p++ 等价于 *p; p+=1;

7.3 指针与函数参数

函数的参数不仅可以是基本类型的变量，也可以是指针变量。对任何必须以地址方式传送的参数，均可以利用指针来实现。特别是函数的返回值多于一个时，可以利用指针来传递函数 的返回值。

注：当指针作为函数的参数时，对应的实参必须是变量的地址或指针；其相应的形 参应是指针变量。这样才能保证地址的正确传送。

【例7.5】将指针作为函数参数，改写［例7.2］的程序。

```
#include 〈stdio.h〉
swap(int *p1,int *p2)
{  int p;
   p=*p1;
   *p1=*p2;
   *p2=p;
}
void main()
{  int a,b;
   int *pa,*pb;
   scanf("%d,%d",&a,&b);
   pa=&a;
   pb=&b;
   if(a<b) swap(pa,pb);
     printf(" \n%d,%d\n",a,b);
}
```

程序运行结果如下：

8，18

18，8