第7章 指针

《C语言程序设计新编教程》

能力要求

CAPACITY

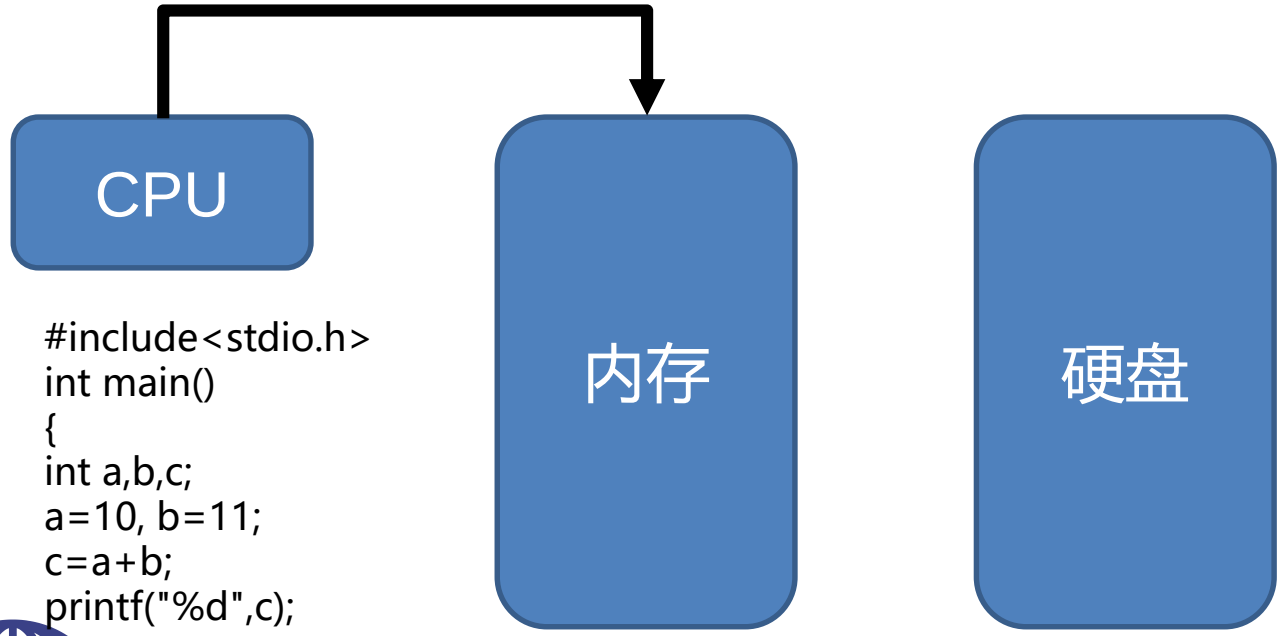
- 掌握C语言指针变量的定义和初始化。
- 理解指针变量的引用。
- 掌握通过指针变量访问数组元素。
- 理解指针变量与字符串的关系。

内容导航

CONTENTS

课程导入

❖变量在内存数据数据区是如何存放的?




```
#include<stdio.h>
int main()
{
int a,b,c;
a=10, b=11;
c=a+b;
printf("%d",c);
}
```

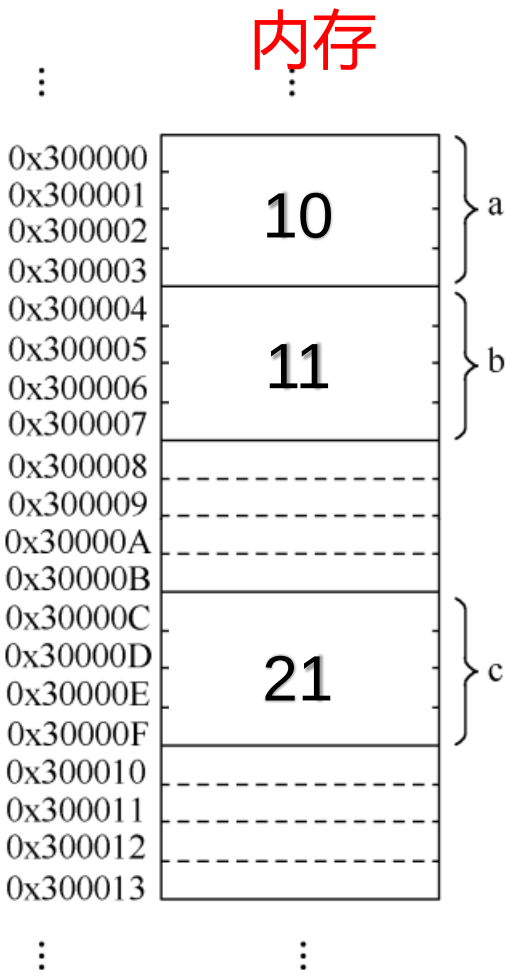


图 6-1 直接访问内存中变量示意图

课程导入

❖**变量在内存数据数据区是如何存放的?**

```
#include<stdio.h>
int main()
{
int a,b,c;
a=10, b=11;
c=a+b;
printf("%d",c);
}
```

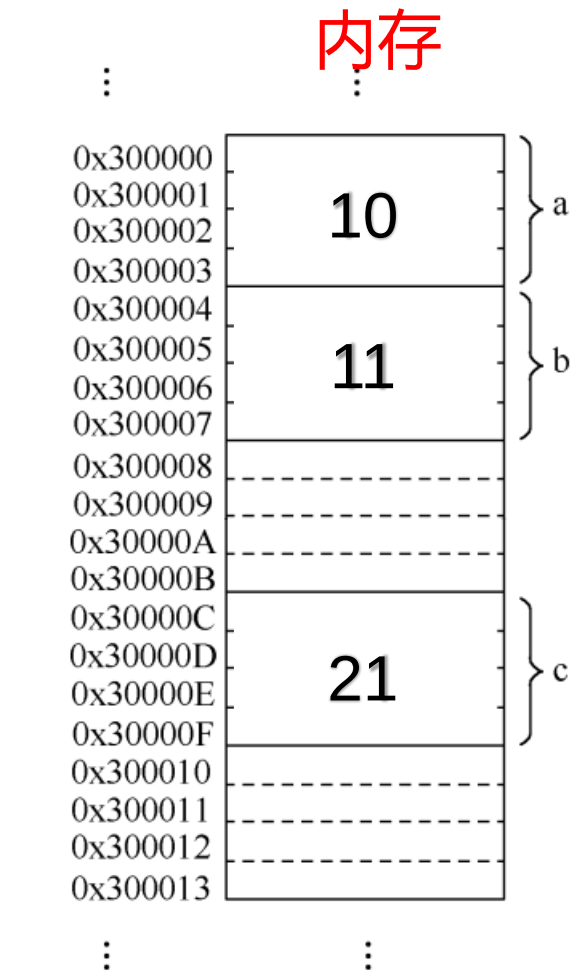


图 6-1 直接访问内存中变量示意图

数据是如何在内存中存放的?

在计算机中，所有的数据都是存放在存储器中的。一般把存储器中的一个字节称为一个内存单元，不同的数据类型所占用的内存单元数是不相同的，例如在int和float占4个字节，double占8个字节，char占1个字节等。

为了正确地访问这些内存单元，必须为每个内存单元编号。根据一个内存单元的编号即可准确地找到该内存单元。

该内存单元的编号也称为地址。也成为指针。

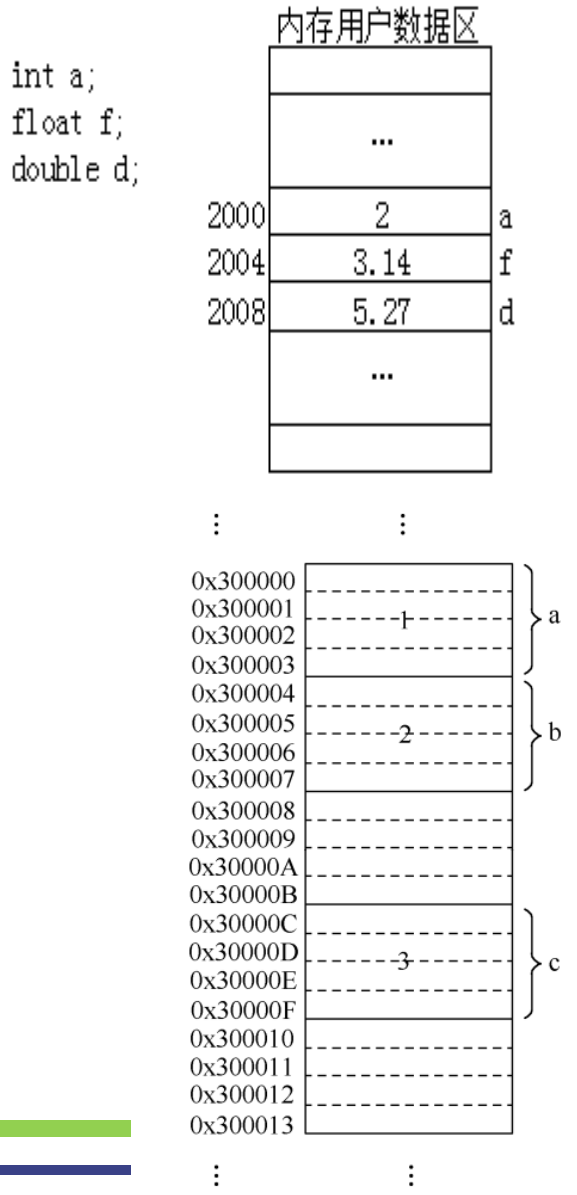


图 6.1 直接访问内存中变量示意图

课程导入

❖变量在内存数据数据区是如何存放的?

```
#include<stdio.h>
int main()
{
int a,b,c;
a=10, b=11;
c=a+b;
printf("%d",c);
}
```

在计算机底层，硬件只认识内存物理地址，

那么计算机如何通过变量名访问到存在内存的数据的呢?

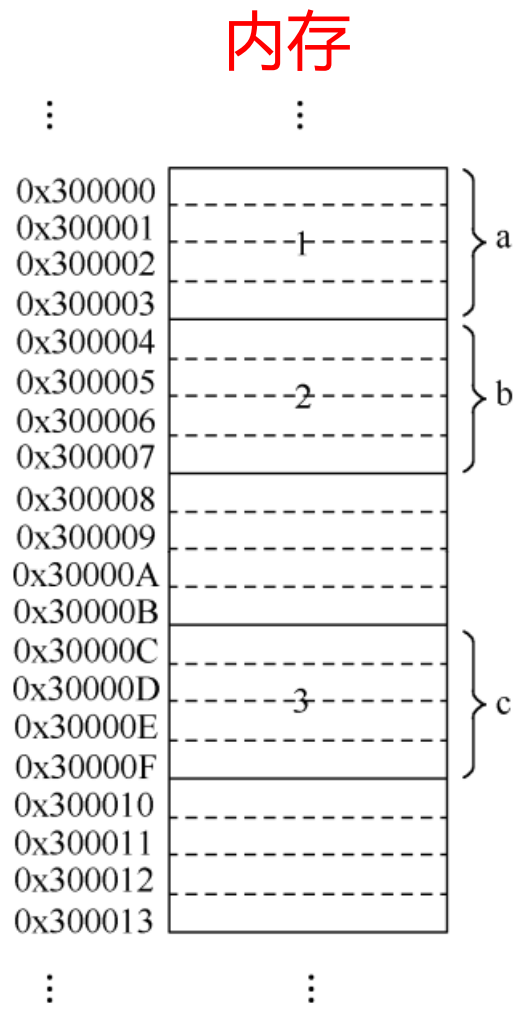


图 6-1 直接访问内存中变量示意图

课程导入

❖变量在内存数据数据区是如何存放的?

```
#include<stdio.h>
int main()
{
int a,b,c;
a=10, b=11;
c=a+b;
printf("%d",c);
}
```

答案是:
通过编译器进行
“变量名”→“地址”的翻译

变量名	起始地址	占据大小
a	0x300000	4
b	0x300004	4
c	0x30000C	4

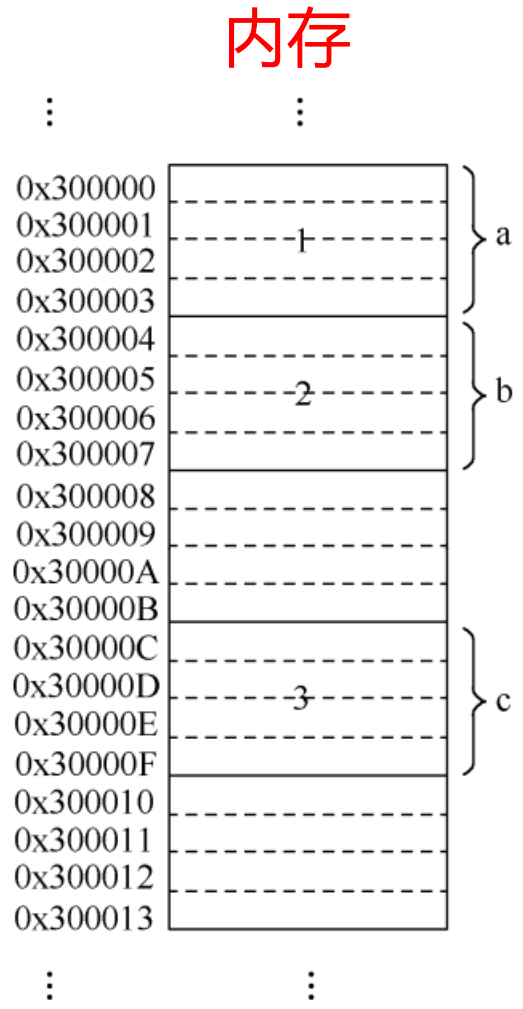


图 6-1 直接访问内存中变量示意图

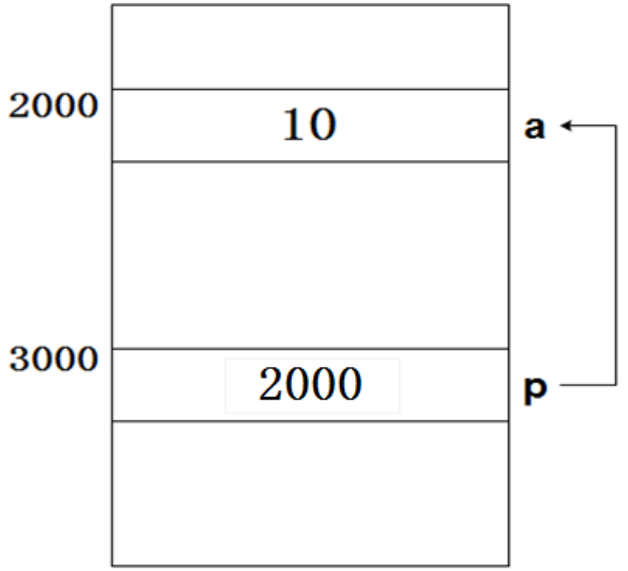
printf(“%d”,a)

scanf(%d,&a)

输出、输入函数的书写规则，谨记

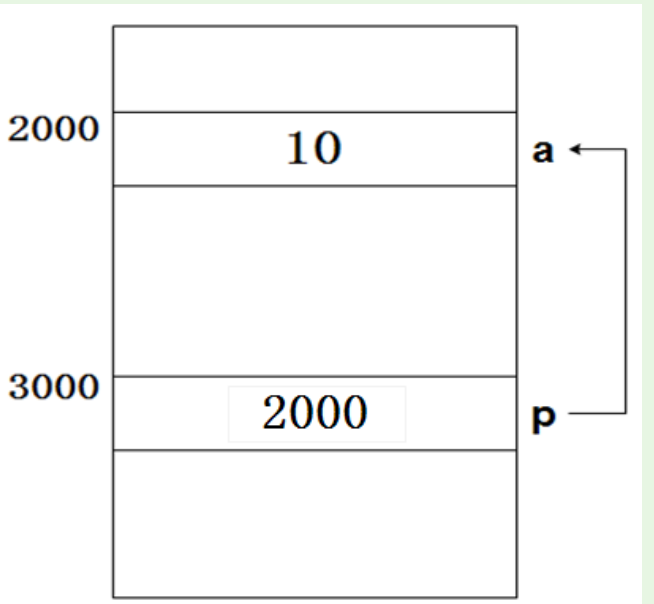
变量的内存地址

- 假设定义整型变量int a=10；系统为该变量分配相应的存储单元，假设其起始地址为2000。
- 通过该地址去访问变量a。在这之前我们是通过变量名来引用变量的值。如：
  printf(“%d”, a);。
- 这种直接按变量名进行的访问，称为“直接访问”方式。



指针的概念

- “间接访问”的方式，即将变量a的地址存放在另一个变量p中，然后通过访问变量p取出p的值，实际上其值为变量a的地址，通过地址访问变量a。
- 一个变量的地址称为该变量的指针。地址2000是变量a的指针，有一种特殊的变量专门用于存放变量的地址（指针），则称它为指针变量。
- 注意区分指针和指针变量这两个概念



大家是不是觉得使用指针很麻烦?

在高阶程序编写，或者设计内存优化的程序编写，是否有用到指针编程?

答案是有的，很多。

在C语言中使用指针有以下几个好处：

1. 节省内存：通过使用指针，可以有效地管理内存，避免内存的浪费。指针可以动态地分配和释放内存，只在需要时才占用内存空间。

2. 提高程序的执行效率：通过使用指针，可以减少对内存的频繁读写操作，提高程序的执行效率。指针可以直接访问内存地址，避免了通过变量名访问内存的过程。

3. 支持动态数据结构：使用指针可以轻松地实现动态数据结构，如链表、树和图等。指针可以简化数据结构的操作，使得数据的插入、删除和查找等操作更加高效。

4. 便于函数间的数据传递：通过使用指针，可以方便地在函数之间传递大块的数据，而不需要复制整个数据。这样可以减少内存占用和提高程序的执行效率。

5. 支持底层操作：C语言中的指针提供了对内存的底层操作，可以直接读写内存地址。这使得C语言可以与硬件进行直接交互，实现底层的操作。

总的来说，指针是C语言中非常重要的概念，它提供了灵活的内存管理和数据操作方式，使得程序更加高效、灵活和可控。但是，指针的使用也需要谨慎，因为不正确的指针操作可能会导致内存错误和程序崩溃。

在C语言中使用指针有以下几个好处：

1. 节省内存：通过使用指针，可以有效地管理内存，避免内存的浪费。指针可以动态地分配和释放内存，只在需要时才占用内存空间。

有效地管理内存举例

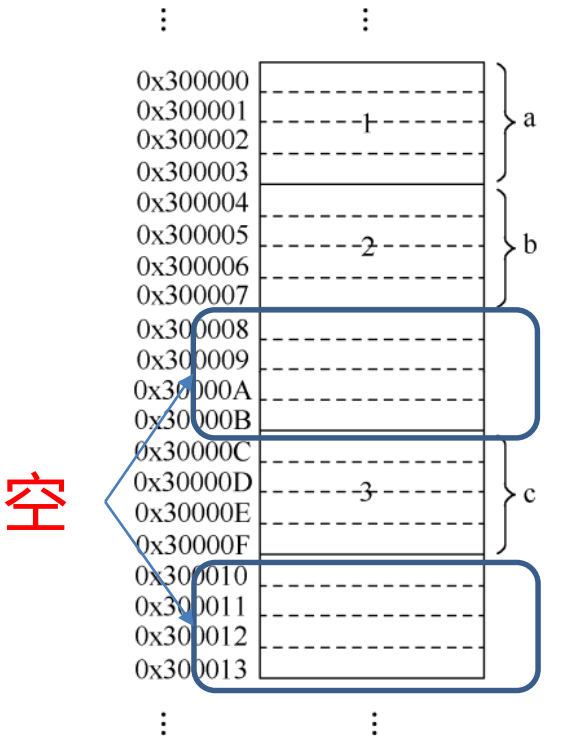


图 6-1 直接访问内存中变量示意图

内容导航

CONTENTS

指针 指针变量

一个变量的使用，有哪几个步骤?

定义，初始化（赋值），引用（使用）

对于指针变量，大家目前学会这几个步骤就可以

指针变量的定义

- 所有变量在使用之前必须要先定义，指定其类型，并按照该类型分配存储单元。定义指针变量的形式为：
- 基类型 *指针变量名；
- int *p;
- double *q;

指针变量是p，不是*p

对指针变量的定义包括：

01 OPTION 指针类型说明，即定义变量为一个指针变量

02 OPTION 指针变量名

03 OPTION 必须指定基类型，指针变量所指向的变量的数据类型

p是指针变量，指针变量不是*p。一个指针变量只能指向同类型的变量

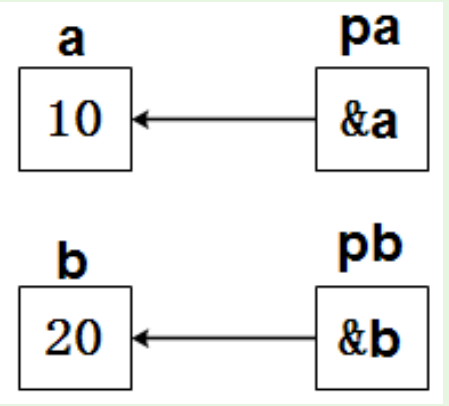
指针变量的引用

- 指针变量只能存放变量的地址，不能将一个非地址类型的数据赋值给一个指针变量。下面的赋值是不合法的。
- int *p;
- p = 123;（p为指向int型的指针变量，不能赋值为123整数。但p=0；除外，表示p为空指针。）
- 两个有关的运算符：
- ①&:取地址运算符。&a是变量a的地址。
- ②*：指针运算符（或称“间接访问” 运算符）。*p代表指针变量p指向的对象。

- 注意：熟练掌握&和*运算符的使用。

【例7.1】通过指针变量访问int型变量

```
#include <stdio.h>
int main()
{
        int a, b;
        int *pa, *pb;
        a = 10;
        b = 20;
        pa = &a;  /*把变量a的地址赋值给pa*/
        pb = &b;  /*把变量b的地址赋值个pb*/
        printf("%d,%d\n", a, b );
        printf("%d,%d\n", *pa, *pb );
}
```




```
10,20
10,20
```

程序说明：

- ①在程序开头处定义了两个指针变量pa和pb，规定它们指向整型变量，但它们并未指向任何一个整型变量。此时的pa和pb的指向是不确定的。在程序中通过pa = &a;和pb = &b;语句赋值，使pa指向a，pb指向b。如图7.2所示。
- ②程序第5、6行的“pa=&a”和 “pb=&b”，是将变量a和b的地址分别赋值给指针变量pa和pb，不能写成“*pa=&a”和 “*pb=&b”。
- ③最后一行使用*pa和*pb用于输出，*pa是指指针变量pa所指向的对象，即*pa和*pb就是变量a和b。最后两个printf函数作用是相同的。
- ④程序中有两处出现*pa和*pb，请区分它们的不同含义。程序的第2行的语句，int *pa, *pb;这里的*表示定义指针变量，而程序最后一行中的*分别表示pa和pb所指向的变量。

指针变量的初始化

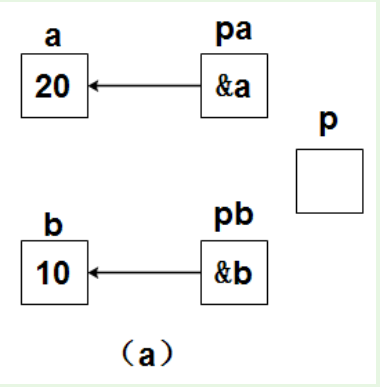
- 程序中经常需要对一些变量预先设置初值，C语言允许指针变量在定义的同时使变量初始化。指针变量初始化的形式：
- 基类型 *指针变量名 = 地址值;
- 说明：地址值可以是变量的地址，数组的首地址，数组元素的地址，执行同类型的指针变量等。
- 下面都是合法的初始化：
    - int a;
    - int *p = &a;
    - int *q = p;

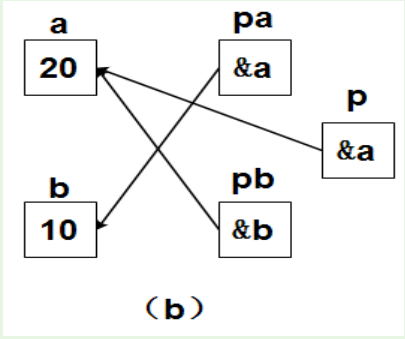
指针变量的运算

- 1．赋值运算
- ① 把一个变量的地址赋值给指向该数据类型的指针变量。
- 例如:
- int a，*pa;
- pa=&a;　/*把整型变量a的地址赋值给指向整型的指针变量pa*/
- ② 把一个指针变量的值赋值给指向相同类型变量的指针变量。
- 例如:
- 　int a，*p=&a,*q;
- 　q=p;　/*把指针变量p的值（地址）赋值给指针变量q，则q也指向变量a*/
- 由于p,q均为指向整型变量的指针变量，因此可以相互赋值。
- ③　把数组的首地址赋值给指向数组的指针变量。
  - 例如:
  - 　int a[10]，*pa;
  - 　pa=a;　/*数组名a代表数组的首地址*/
  - 　pa=&a[0];　/*数组第一个元素的地址也是整个数组的首地址*/

【例7.2】输入两个整数，按由小到大的顺序输出

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

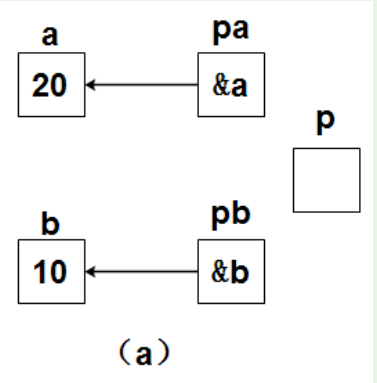




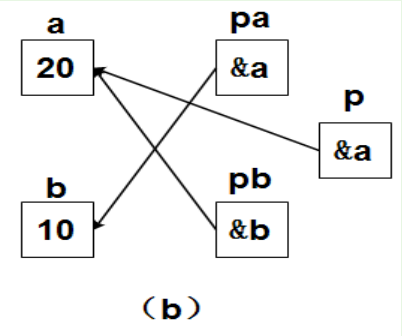
【例7.2】输入两个整数，按由小到大的顺序输出

程序运行结果：

```
20 10
a = 20,b = 10
*pa = 20,*pb = 10
a = 20,b = 10
*pa = 10,*pb = 20
```



（a）



（b）

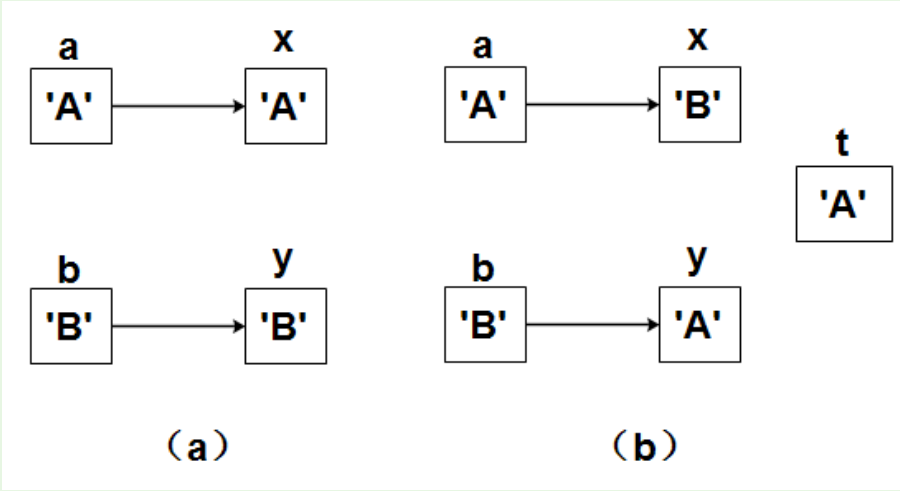
【例7.2】输入两个整数，按由小到大的顺序输出

- 程序说明：
  - ①程序中定义了两个int型变量a和b。另外定义三个指向int型的指针变量。
  - ②程序中通过语句pa = &a;pb = &b;分别是pa指向a，pb指向b。
  - ③通过语句scanf("%d%d", pa, pb);输入数据保存到pa和pb所指向的存储单元中，即使变量a和b有确定的值。
  - ④下面两printf语句，分别输出变量a和b的值及指针变量pa和pb所指向单元中的数据，输出结果相同。
  - ⑤通过if语句判断a和b的值，如果a大于b，进行交互指针变量pa和pb的指向，而int型变量a和b的值不变。
  - ⑥上面的if语句可以改写成if(*pa>*pb)，效果一样。

指针变量作为函数参数

```
int sub1( char x, char y )
{
        char t;
        t = x; x = y; y = t;
}
```

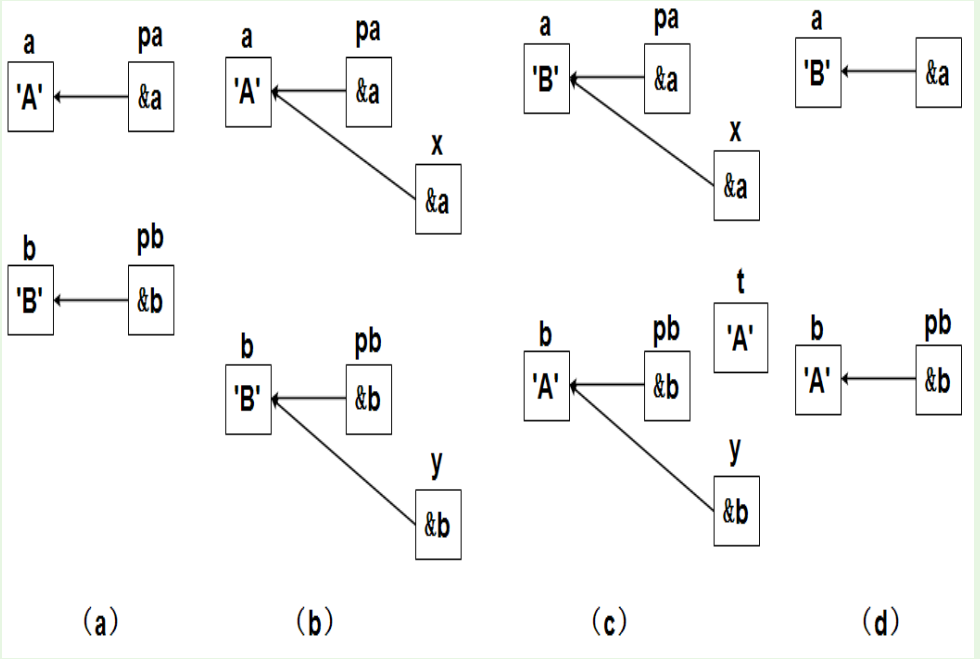
程序说明：
在main函数中，通过语句sub1(a, b);调用用户自定义函数sub1。函数调用时实参分别是int型变量a, b。值传递。将实参a的值赋值给形参x，将实参b的值赋值给形参y。
sub1函数完成变量x和y的交换，不影响实参变量a和b。



指针变量作为函数参数

```
void sub2( char *x, char *y )
{
        char t;
        t = *x; *x = *y; *y = t;
}
```

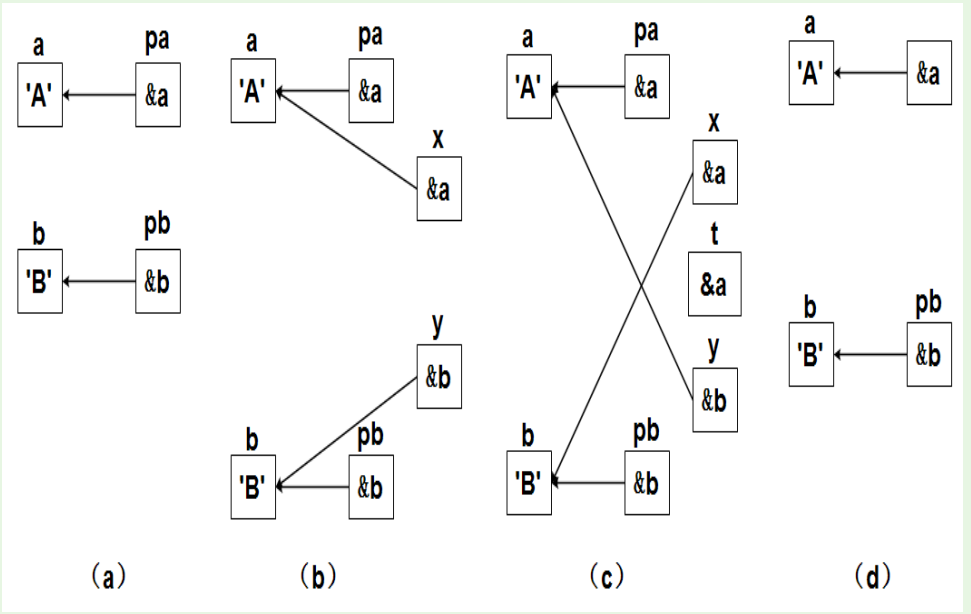
程序说明：
在main函数中，通过语句sub2(pa, pb);调用用户自定义函数sub2。函数调用时实参分别是int型变量a, b的地址。sub2函数调用结束后，变量a和b交换后的值被保留下来。



指针变量作为函数参数

```
void sub3( char *x, char *y )
{
        char *t;
        t = x; x = y; y = t;
}
```

程序说明：
在main函数中，通过语句sub3(pa, pb);调用用户自定义函数sub3。函数调用时实参分别是int型变量a, b的地址。在用户自定义函数sub3中完成了两个形参指针x和y的交换，不没有交换形参x和y所指向单元中的内容。sub3函数调用结束后，变量a和b不受影响，没有交换。



【例7.3】指针变量做函数参数

```
#include "stdio.h"
void sub1( char x, char y )
{
        char t;
        t = x; x = y; y = t;
}
void sub2( char *x, char *y )
{
        char t;
        t = *x; *x = *y; *y = t;
}
void sub3( char *x, char *y )
{
        char *t;
        t = x; x = y; y = t;
}
```

```
void main()
{
    char a, b;
    char *pa, *pb;
a = 'A'; b = 'B'; sub1( a, b ); printf("sub1--%c%c\n", a,
b );
a = 'A'; b = 'B'; pa=&a; pb=&b; sub2( pa, pb );
printf("sub2--%c%c\n", a, b );
a = 'A'; b = 'B'; pa=&a; pb=&b; sub3( pa, pb );
printf("sub3--%c%c\n", a, b );
}
```

程序运行结果：

```
sub1--AB
sub2--BA
sub3--AB
```

- 指针变量有两个运算符： “&”（取地址运算符）和“*” （指针运算符）

- （1）取地址运算符&
- 取地址运算符“&”将一个变量的地址（指针）赋给一个指针变量， “&”表示取变量的地址，一般形式为： “&变量名;”，如“&a”表示变量a的地址。下列语句将变量i的地址赋给指针变量p。

```
int i, *p;
i=3;
p=&i;
```

指针变量的赋值只能赋予地址，决不能赋予任何其他数据，否则将引起错误.

如将整数值0x300000赋给指针变量 “p=0x300000;”是非法的。变量的地址是由编译器分配的，对用户完全透明，用户不知道变量的具体地址。

- （2）指针运算符*

- 指针运算符“*”用来存取指针变量所指向的目标变量的值。

- 如以上代码示例中，“*p”即为i。取地址运算符&和指针运算符*均为单目运算符。

```
int i, *p;
i=3;
p=&i;
```

- 运算符“&”用来取目标变量的地址，运算符“*”用来取指针所指向目标变量的内容，两种互逆运算。当“*&”或“&*”在一起时，具有抵消作用。

- 例如，“*ptr_i=*&I;”相当于“*ptr_i=I;”。
- 定义int型指针变量ptr_i，并指向int型变量i，则“ptr_i”为指针变量，它的内容是地址量；“*ptr_i”为指针的目标变量，它的内容是数据；“&ptr_i”为指针变量占用内存的地址。

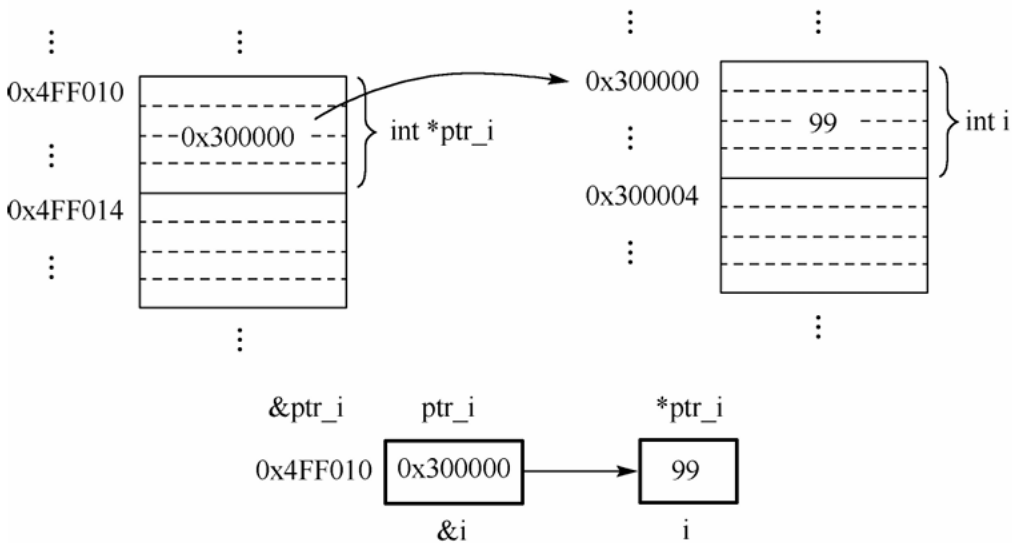


图 6-4 运算符 & 与 * 关系示意图

```
#include<stdio.h>
int main()
{
int i, *p;
i=3;
p=&i;



printf("%d",i);
}
```

```
#include<stdio.h>
int main()
{
int i, *p;
i=3;
p=&i;
*p=4;



printf("%d",i);
}
```

打印的结果是什么?

各种数据类型的存储大小

```
char        //字符数据类型----1个字节
short       //短整型-----2个字节
int         //整形----4个字节
long        //长整型----大于或者整型的字节----也就是4或者8个字节
            //不建议使用long类型，有的平台下显示是4个字节，有的平台下显示是8个字节
long long   //更长的整形----8个字节
float       //单精度浮点数----4个字节
double      //双精度浮点数-----8个字节
```

一个内存单元1字节，8比特

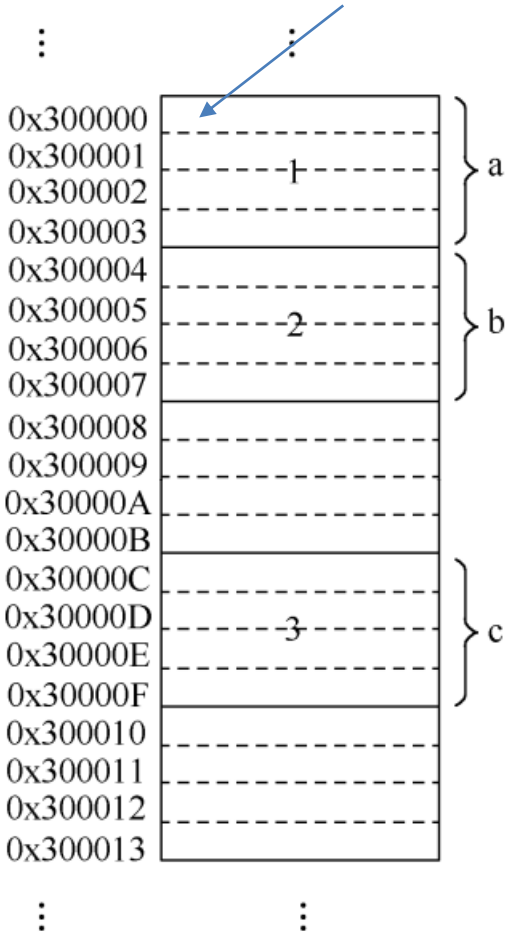


图 6-1 直接访问内存中变量示意图

int型数据大小验证

```
#include<stdio.h>
int main()
{
		int *o;
		int *p;
		int *q;

		int a;
		int b;
		int c;
		a=1;
		b=2;
		c=3;


		o=&a;
		p=&b;
		q=&c;
		printf ("%p\n",o);
		printf ("%p\n",p);
		printf ("%p",q);
}
```

```
000000000062FE04
000000000062FE00
000000000062FDFC
--------------------------------

Process exited after 0.1653 seconds with return value 0
请按任意键继续. . .
```

short型数据大小验证

```
#include<stdio.h>
int main()
{
		short *o;
		short *p;
		short *q;

		short a;
		short b;
		short c;
		a=1;
		b=2;
		c=3;


		o=&a;
		p=&b;
		q=&c;
		printf ("%p\n",o);
		printf ("%p\n",p);
		printf ("%p",q);
}
```

```
000000000062FE06
000000000062FE04
000000000062FE02
--------------------------------

Process exited after 0.1762 seconds with return value 0
请按任意键继续. . .
```

```
#include<stdio.h>
int main()
{
        int *o;
        int *p;
        int *q;

        short a;
        short b;
        short c;
        a=1;
        b=2;
        c=3;


        o=&a;
        p=&b;
        q=&c;
        printf ("%d\n",o);
        printf ("%d\n",p);
        printf ("%p",q);
}
```

大家一起来找茬~

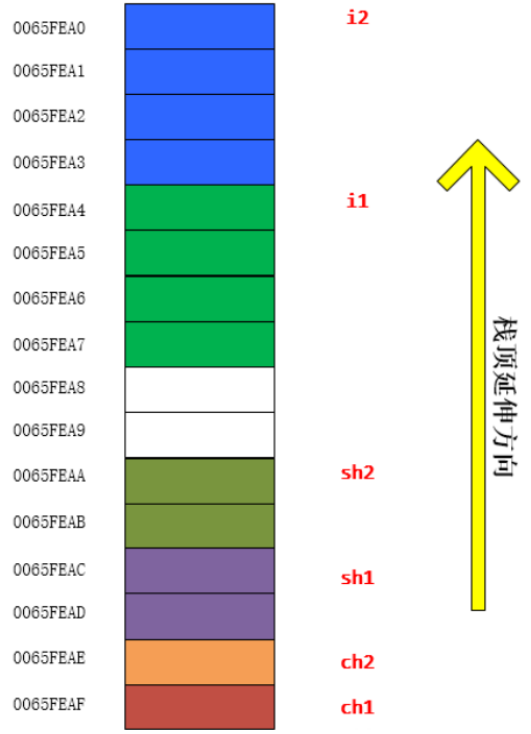
这段代码哪里有错?

2处大家会因为粗心而写出的地方

```
#include<stdio.h>
int main()
{
        int *o;
        int *p;
        int *q;

        int a;
        int b;
        int c;
        a=1;
        b=2;
        c=3;


        o=&a;
        p=&b;
        q=&c;
        printf ("%d\n",o);
        printf ("%d\n",p);
        printf ("%d",q);
}
```

一、程序中的栈

- 栈是现代计算机程序里最为重要的概念之一
- 栈在程序中用于维护函数调用上下文
- 函数中的参数和局部变量存储在栈上




```
000000000062FE04
000000000062FE00
000000000062FDFC
--------------------------------

Process exited after 0.1653 seconds with return value 0
请按任意键继续. . .
```

课外知识增长：

为什么编程语言中需要堆和栈
https://blog.csdn.net/u010632868/article/details/79465569

C语言中的栈、堆和静态存储区
https://blog.csdn.net/weixin_43129713/article/details/124093704

- 多重指针
- 如果一个指针变量中存放的是另一个指针变量的地址，则称这个指针变量为指向指针的指针变量 （或者称为两重指针或两级指针）。通过指向指针的指针变量来访问变量，形成“二级间址”。 定义一个两重指针的形式为“类型说明符 **指针变量名;”，指针变量名前有两个“*”，相当于 “*(*指针变量名)”，表示该指针变量是指向一个指针型变量的。如“int **pp;”定义一个两重指针pp， 它指向另外一个指针变量p，而指针变量p指向一个int型变量，

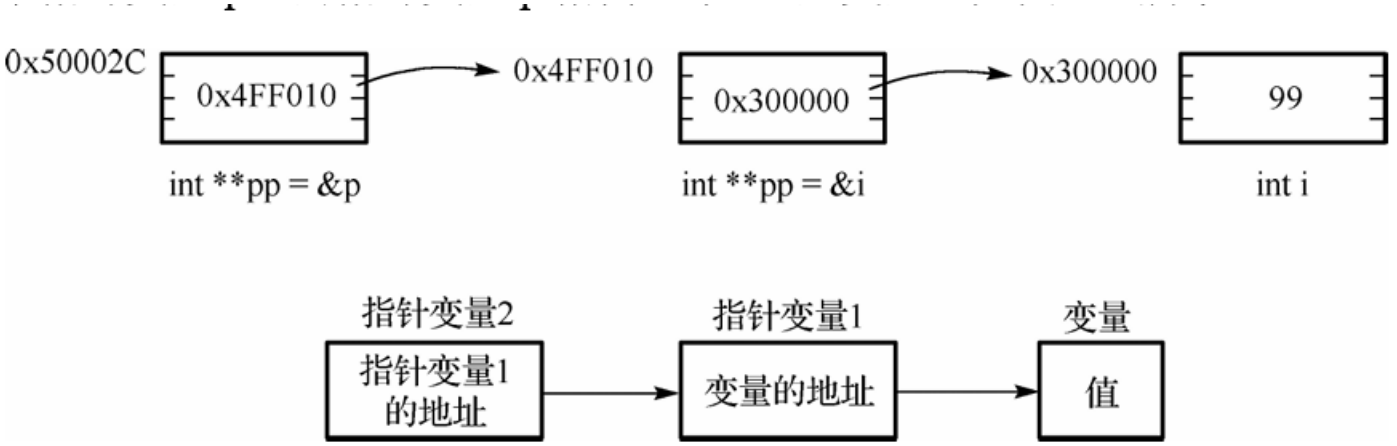


图 6-6 两重指针

```
#include<stdio.h>
int main()
{
        int i=99;
        int *p=&i;
        int **pp=&p;
        int ***ppp=&pp;
        int ****pppp=&ppp;

        printf ("%p\n",p);
        printf ("%p\n",pp);
        printf ("%p\n",ppp);
        printf ("%p\n",pppp);
}
```

内容导航

CONTENTS

指针与一维数组

```
#include<stdio.h>
int main()
{
        int c;
        int b;
        int a;
        a=1;
        b=2;
        c=3;
}
```

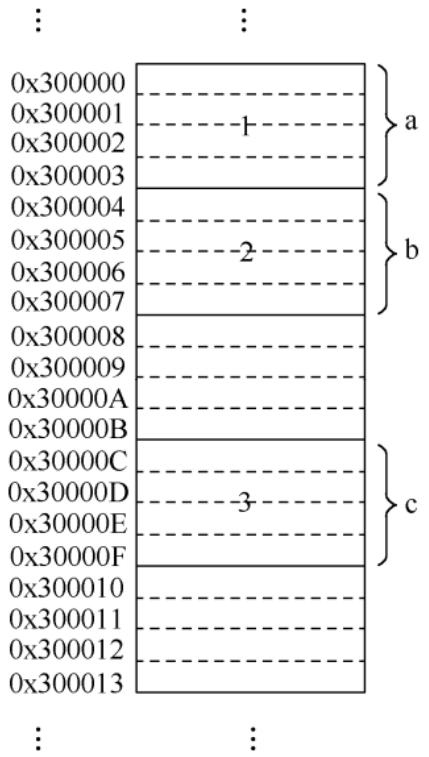


图 6-1 直接访问内存中变量示意图

一维数组呢?

int array[10];

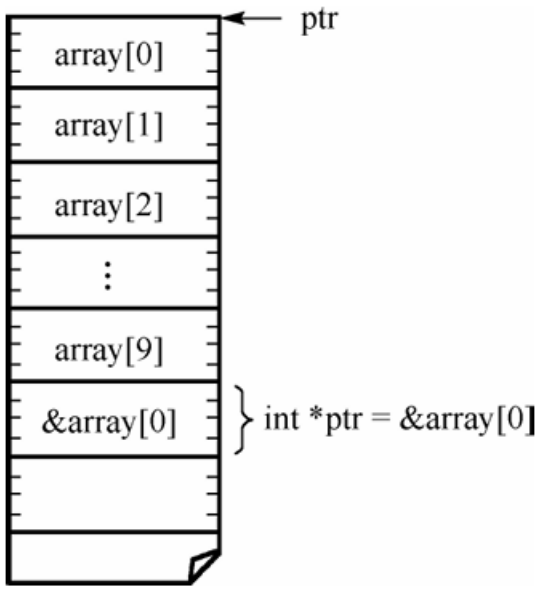


图 6-7 指向一维数组元素的指针变量

指针与一维数组

- 1．指向一维数组的指针变量的定义及赋值
- 定义方法和指向变量的指针定义相同。赋值方法如下：
- ①定义的同时赋初值：
- int a[10], *p = a;
- ②先定义，后赋值：
- int a[10], *p;
- p = a;
- 说明：
- C语言规定，数组名是数组的首地址，也即元素a[0]的地址。因此下面两条语句等价：
    - p = a; 和 p = &a[0];

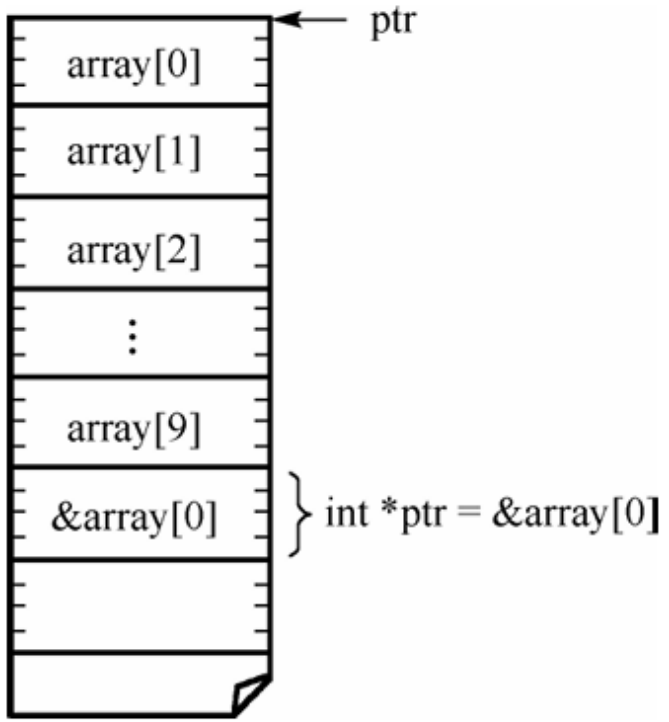


图 6-7　指向一维数组元素的指针变量

```
#include<stdio.h>
int main()
{
int array[4]={1,2,3,4};

int *a;
int *b;
int *c;
int *d;

a=&array[0];
b=&array[1];
c=&array[2];
d=&array[3];

                printf ("%p\n",a);
                printf ("%p\n",b);
                printf ("%p\n",c);
                printf ("%p\n",d);

}
```

```
00000000062FDE0
00000000062FDE4
00000000062FDE8
00000000062FDEC
```

指针与一维数组

- 2. 一维数组元素地址和值的表示方法

方法	地址	值	
下标法	&a[i]	a[i]	
指针法（地址法）	a+i	*(a+i)	数组名+数字
	p+i	*(p+i)	地址+数字

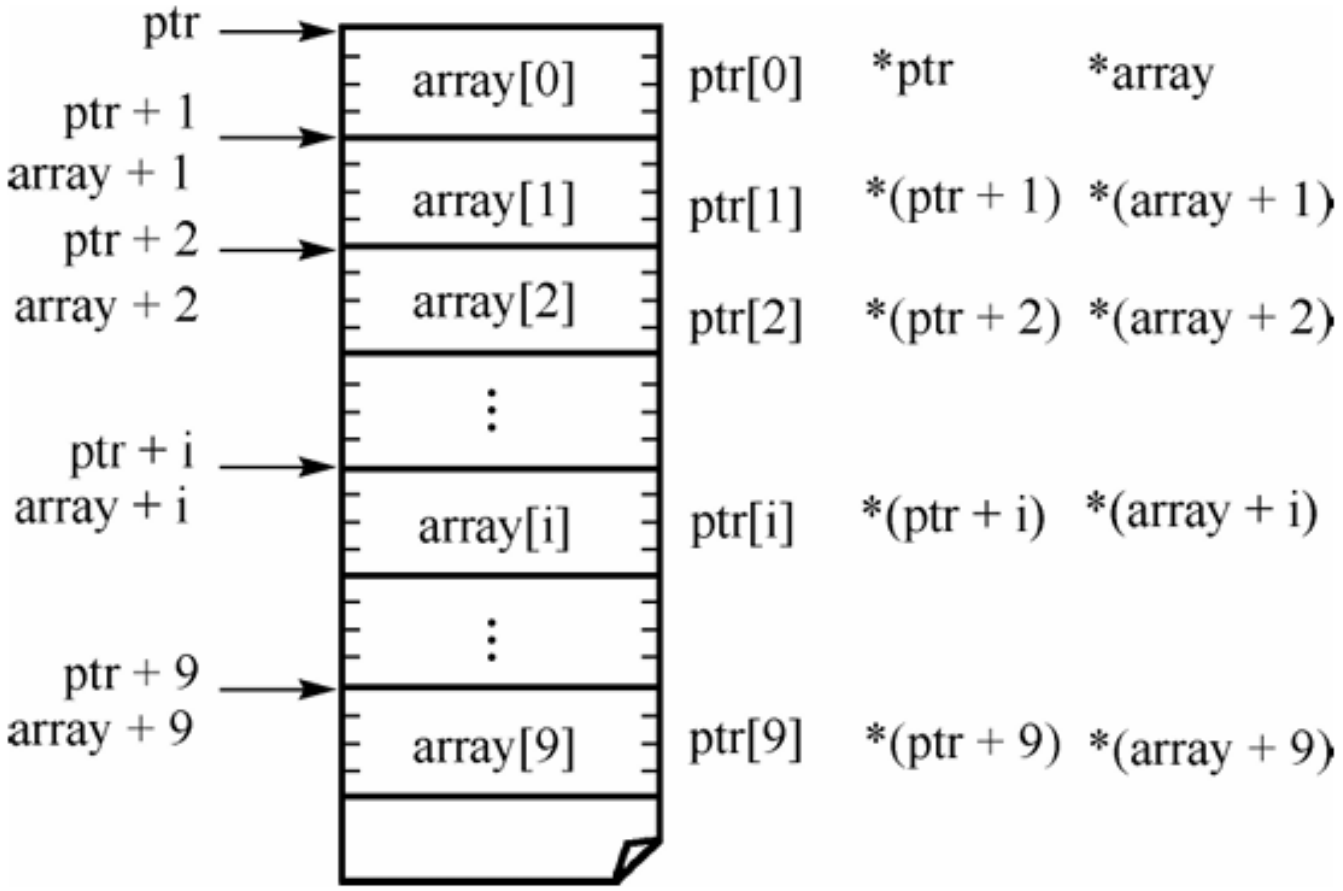


图 6-8　数组元素的表示方法

数组名+数字

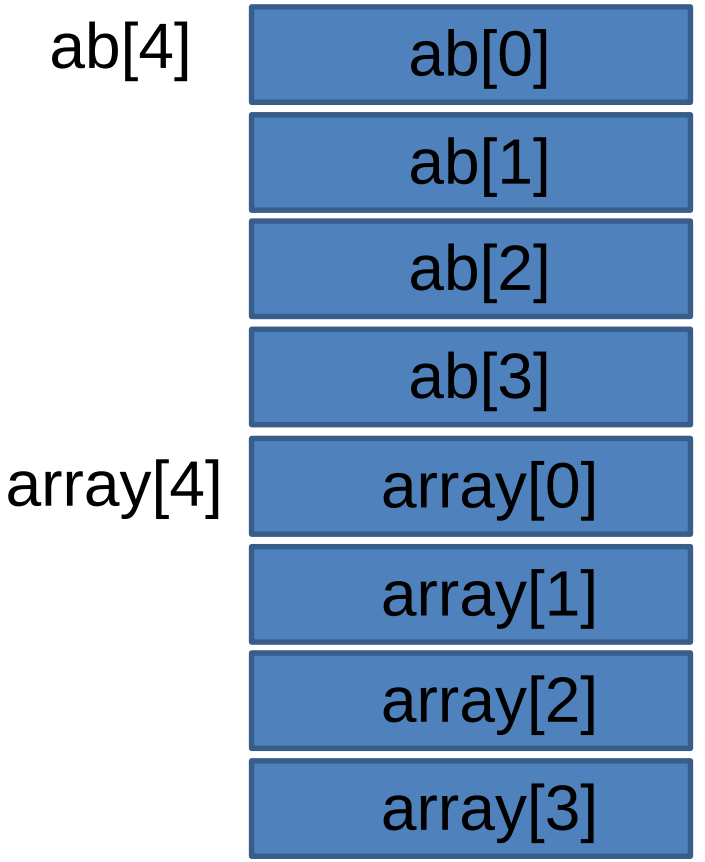
```
#include<stdio.h>
int main()
{
int array[4]={1,2,3,4};

int *a;
int *b;
int *c;
int *d;

a=&array[0];
b=&array[1];
c=&array[2];
d=&array[3];

        printf ("%d\n",*(array+3));
}
```

地址+数字

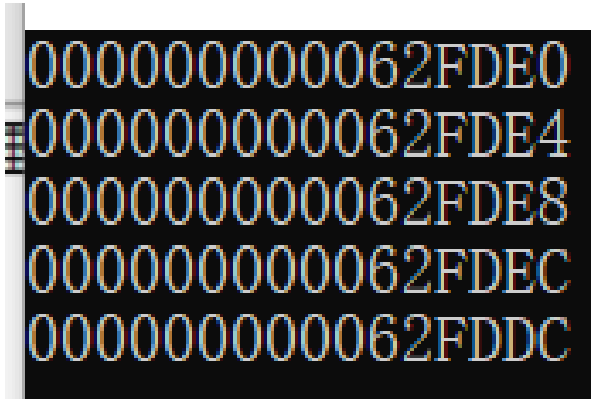
```
#include<stdio.h>
int main()
{
int array[4]={1,2,3,4};

int *a;
int *b;
int *c;
int *d;

a=&array[0];
b=&array[1];
c=&array[2];
d=&array[3];

        printf ("%d\n",*(a+1));
}
```

多个数组的话，数组之间的地址是如何排列的?

```
int array[4]={1,2,3,4};
int ab[4]={1,2,3,4};
```



```
int array[4]={1,2,3,4};
int ab[4]={1,2,3,4};
int *a;
int *b;
int *c;
int *d;
int *e;

a=&array[0];
b=&array[1];
c=&array[2];
d=&array[3];
e=&ab[3];

        printf ("%p\n",a);
        printf ("%p\n",b);
        printf ("%p\n",c);
        printf ("%p\n",d);
        printf ("%p\n",e);
}
```



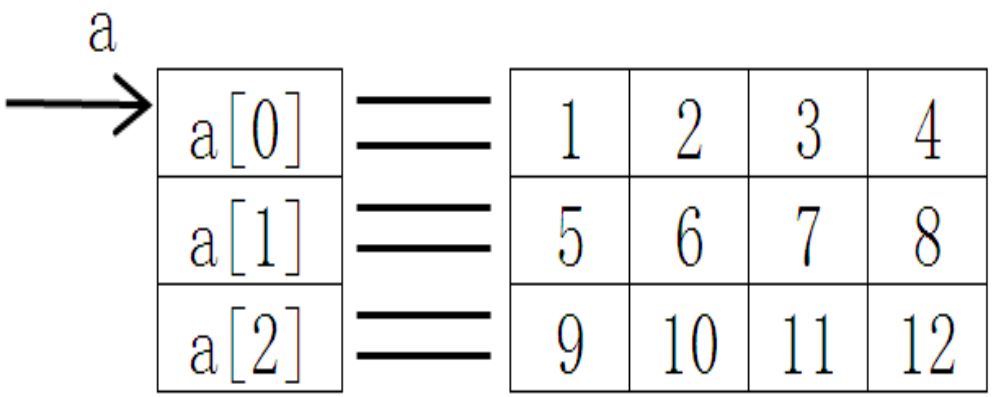
指针与二维数组

1. 二维数组元素的地址

首先复习二维数组的性质。

int a[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };

定义二维数组a，含有3行4列，共12个元素。



指针与二维数组

1. 二维数组元素的地址

从二维数组的角度出发，a数组名，代表二维数组的首地址，是所对应的首元素a[0]的地址即&a[0]，a+1指向下一个元素，即a[1]的地址&a[1]，是二维数组a第一行的地址。如果二维数组a的首地址为2000，在vc6.0中int型占4字节，则a+1为2016。

a (2000)	1	2	3	4
a+1 (2016)	5	6	7	8
a+2 (2032)	9	10	11	12

指针与二维数组

表示形式	含义	地址
a	二维数组名，指向一维数组a[0],即0行首地址	2000
a[0],*a(+0),*a	0行0列元素的地址	2000
a+1, &a[1]	1行首地址	2016
a[1],*a+1	1行0列元素a[1][0]的地址	2016
a[1]+2,*(a+1)+2,&a[1][2]	1行2列元素a[1][2]的地址	2024
(a[1]+2),(*(a+1)+2),a[1][2]	1行2列元素a[1][2]的值	元素值为7

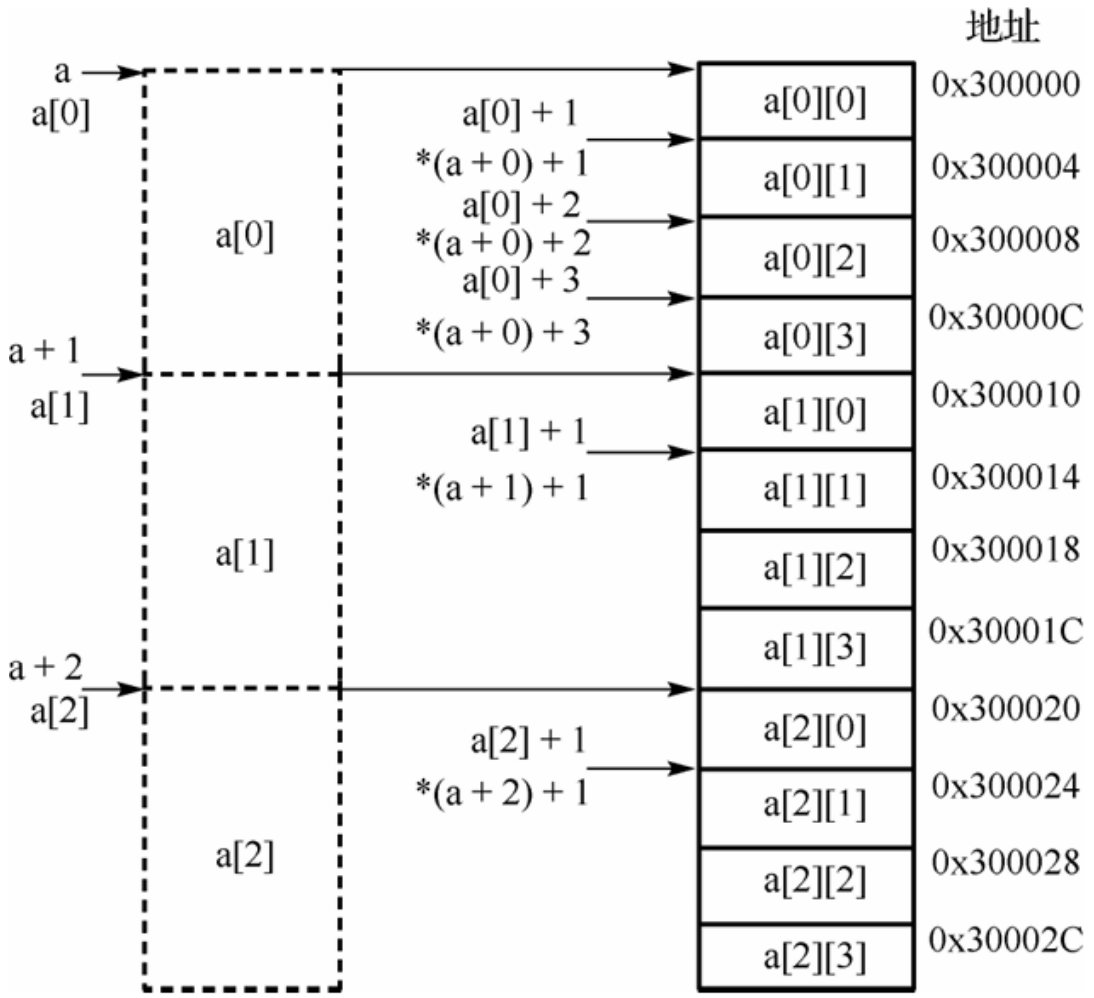


图 6-9　二维数组元素的地址

内容导航

CONTENTS

字符数组元素的引用

根据字符数组的下标引用字符数组中的元素，得到一个字符。

【例5.4】输入一行字符串

```
char str[15]={‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘!’, ‘W’,’o’ ,‘r’, ‘l’, ‘d’, ‘!’};
```

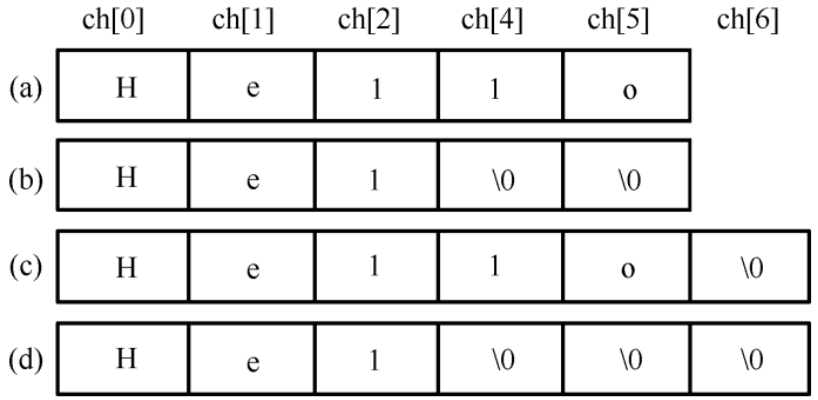


图 5-5 字符数组初始化示意图

程序代码：

```
/*ex5_4.C:输出一行字符串*/
#include<stdio.h>
int main()
{
    int i;
char str[15]={‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘!’, ‘W’,’o’ ,‘r’, ‘l’, ‘d’, ‘!’};
    for(i=0;i<15;i++)
      printf("%c",str[i]);
    printf("\n");
}
```

指针访问字符串

- 字符数组可以用来存放字符串，使用指向字符类型的指针变量也可以访问字符串。
- 用指针变量访问字符串分两步：

01 OPTION ① 先定义一个指向字符型的指针变量

02 OPTION ② 将一个字符串的首地址赋值给该指针变量。

【例7.10】字符指针的定义和使用

```
#include "stdio.h"
int main()
{
        char *p = "This is a string";
        printf("%s\n", p );
}
```

P中存放的是字符串在内存中的首地址

程序运行结果：

```
This is a string
```

①C语言对字符串常量是按字符数组处理的，在内存中开辟一段连续的存储单元。char *p = "This is a string";该语句是在定义指针变量的同时给p初始化，实际上是将字符串在内存中的首地址赋给指针变量p，而不是把一个字符串赋给指针变量。

②char *p = "This is a string";该语句等价于

char *p; p = “This is a string”;

【例7.10】字符指针的定义和使用

```
#include "stdio.h"
int main()
{
        char *p = "This is a string";
        printf("%s\n", p );
}
```

P中存放的是字符串在内存中的首地址

程序运行结果：

```
This is a string
```

程序到此吧！

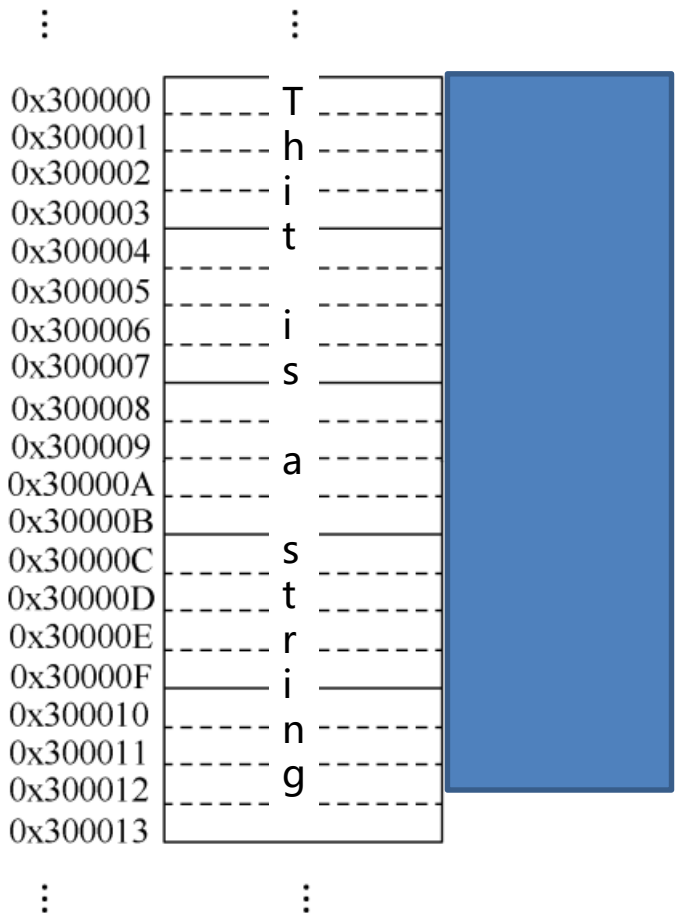


图 6-1 直接访问内存中变量示意图

使用字符串指针变量与字符数组的区别

用字符数组和字符指针变量都可实现字符串的存储和运算。但是两者是有区别的。在使用时应注意以下几个问题：

①存储形式不同。字符串指针变量本身是一个变量，用于存放字符串的首地址。而字符串本身是存放在以该首地址为首的一块连续的内存空间中并以 ‘\0’作为串的结束。字符数组是由于若干个数组元素组成的，它可用来存放整个字符串。

②初始化形式不同。对指向字符类型的指针变量初始化形式：

char *p = “This is a string”;

对字符数组的初始化形式：

char s[]={"This is a string"};

使用字符串指针变量与字符数组的区别

③赋值方式不同。对指向字符类型的指针变量可以赋值，如：

char *p; p = “This is a string”;

而对与字符数组只能逐个元素赋值，如：

char s[20]; for(i=0; i<20; i++) s[i] = ‘a’;

不能写为：char s[20]; s={"This is a string"};

S是数组名为地址常量，不能给常量赋值。只能在数组定义时整体赋值，或者使用strcpy(s, “This is a string”);不能在赋值语句中整体赋值。

使用字符串指针变量与字符数组的区别

④字串的输入。定义一个字符数组，在编译时分配存储单元，有确定的地址。

如： char st[20]; 可以使用scanf(“%s”, st);输入字符串。而定义指向字符类型的指针变量，如 char *p;只是给指针变量分配存储单元，但是p没有确切的指向，使用scanf(“%s”, p);是错误的。

```
#include<stdio.h>
int main()
{
    int i;
char str[]={"Hello!"};
    for(i=0;i<6;i++)
      printf("%c",str[i]);
    printf("\n");
}
```

```
Hello!
```

```
#include <stdio.h>
int main()
{
char *p = "Hello!";
printf("%s\n", p+2 );
}程序运行结果：
```

```
Hello!
```

【例7.11】指向字符类型的指针变量。

```
#include "stdio.h"
void main()
{
        char *p = "01234567";
        p += 3;
        printf("%c,%s", *p, p );
        printf("\n");
}
```

程序运行结果：

```
3,34567
```

①定义指向字符类型的指针变量p，用于存放字符串“01234567”在内存中的首地址。不是存放字符串的值。
②P+=3；使指针变量p指向元素3。
③使用%c格式字符，输出指针变量所指向的元素，结果为3。使用%s格式字符输出从指针变量p所指的元素开始往后输出直到遇见’\0’结束。结果为34567。
⑥用指针变量指向一个格式字符串。可以代替printf函数中的格式字符串。如：

```
int a = 10, b = 20;
        char *p = "a=%d,b=%d\n";
        printf( p, a, b );
```

- 1．用指针方法实现：输入两个整数，输出其中较大的数。
  2．用指针方法实现：输入三个整数，将这三个数按从小到大的顺序输出。
  3．用指针方法实现：计算一个数组中所有元素的和及平均值。
  4．用指针方法实现：输入一个二维数组，再输入行号和列号，输出对应的数值。
  5．用指针方法实现：输入一串字符，计算其中空格的个数。


THANKS