西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

C程序设计

Programming in C

1011014

主讲：姜学锋，计算机学院

批量文字简洁表示

1、指向字符串的指针

2、指向字符串数组的指针

7.4 指针与字符串

- 可以利用一个指向字符型的指针处理字符数组和字符串，其过程与通过指针访问数组元素相同。使用指针可以简化字符串的处理，是程序员处理字符串常用的编程方法。

7.4.1 指向字符串的指针

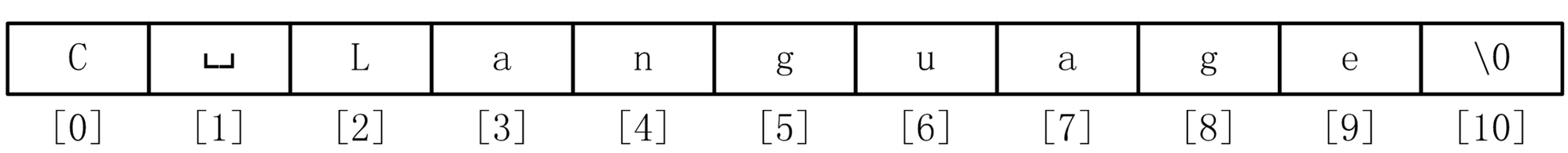
- 可以定义一个字符数组，用字符串常量初始化它，例如：

```
char str[]="C Language";
```

- 如图所示。系统会在内存中创建一个字符数组str，且将字符串常量的内容复制到数组中，并在字符串末尾自动增加一个结束符'\0'。

7.4.1 指向字符串的指针

图7.18 字符串的数组形式



7.4.1 指向字符串的指针

- C语言允许定义一个字符指针，初始化时指向一个字符串常量，一般形式为：

```
char *字符指针变量=字符串常量, ......
```

7.4.1 指向字符串的指针

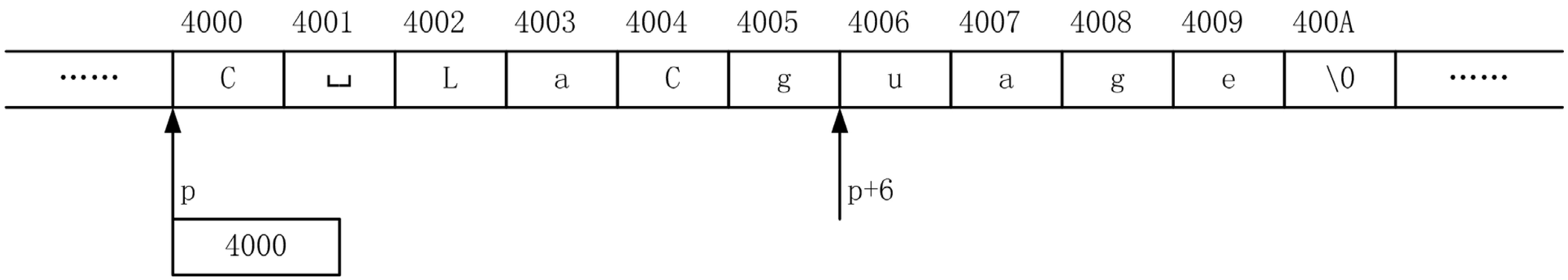
- 例如：

```
char *p="C Language";
```

- p是一个指向char型的指针变量。

7.4.1 指向字符串的指针

图7.19 指向字符串的指针



如图所示。这里虽然没有定义字符数组，但在程序全局数据区中仍为字符串常量分配了存储空间，而且以数组形式并在字符串末尾自动增加一个结束符'\0'。显然，这个字符串常量是有地址的。初始化时，p存储了这个字符串首字符地址4000，而不是字符串常量本身，称p指向字符串。

7.4.1 指向字符串的指针

- 还可以在程序语句中，用字符串常量赋值给字符指针变量p。例如：

```
char *p;
p="C Language"; //正确 "..."字符串常量既是字符数组，又表示字符串首地址，两者均是char*
```

- 无论哪种形式都是为指针变量赋地址值，而不是对*p赋值。赋值过程中只是将字符串的首地址值存储在p中，而不是将字符串存储在p中。p仅是一个指针变量，它不能用来存放字符串的全部字符，只能用来存放一个字符串的指针（或地址）。

7.4.1 指向字符串的指针

- 字符指针变量p除指向字符串常量外，还可以指向字符数组。例如：

```
char str[]="C Language", *p=str; //p指向字符串的指针
```

- 通过字符指针可以访问字符串，例如通过字符指针输出字符串。

```
printf("%s", p);
```

7.4.1 指向字符串的指针

- %s格式会将输出项当作字符串输出，输出项参数此时必须为字符串地址，printf从该地址对应的字符开始输出，每次地址自增，直到遇到空字符为止。例如：

```
char str[]="C Language", *p=str; //p指向字符串的指针
printf("%s\n", p); //输出：C Language
printf("%s\n", p+2); //输出：Language
printf("%s\n", &str[7]); //输出：age
```

7.4.1 指向字符串的指针

- 下面是通过字符指针遍历字符串的两段代码。
- 程序①：

```
char str[]="C Language", *p=str; //p指向字符串的指针
while (*p!='\0') printf("%c",*p++);
```

- 程序②：

```
char *p="C Language"; //p指向字符串常量的指针
while (*p!='\0') printf("%c",*p++);
```

7.4.1 指向字符串的指针

- 两段程序的运行结果相同，但它们之间有一个重要区别：即记忆字符串首地址的方式不一样。程序①运行后若要让p再次指向字符串，只要p=str即可，因为字符串的首地址就是字符数组名。而程序②运行后若要让p再次指向字符串，就困难了。因为字符串的首地址开始给了p，但运算p++后，p发生了变化从而使得p变成了“迷途指针”。

```
char *p="C Language"; //p指向字符串常量的指针
while (*p!='\0') printf("%c",*p++);
```

7.4.1 指向字符串的指针

- 解决这个问题的办法是在程序②中另外引入一个指针变量记住字符串的首地址。例如：

```
char *p1, *p="C Language"; //p指向字符串的首地址
p1=p; //p变化前先将字符串的首地址保存到p1中
while (*p1!='\0') printf("%c",*p1++); //p1修改而p保持不变
```

7.4.1 指向字符串的指针

【例7.15】

编写程序计算一个字符串的长度（实现strlen函数的功能）。

7.4.1 指向字符串的指针

例7.15

```
1 #include <stdio.h>
2 int main()
3 {
4   char str[100],*p=str;
5   scanf("%s", str); //输入字符串
6   while (*p) p++; //指针p指向到字符串结束符
7   printf("strlen=%d\n",p-str); //输出字符串长度
8   return 0;
9 }
```

程序第6行while表达式的*p是*p!='\0'的简写形式，两者作为逻辑结果是完全等价的，含义是判断p所指向的数组元素是否为空字符'\0'；如果不为空字符则p++，使指针移向下一个元素继续判断。

7.4.1 指向字符串的指针

例7.15

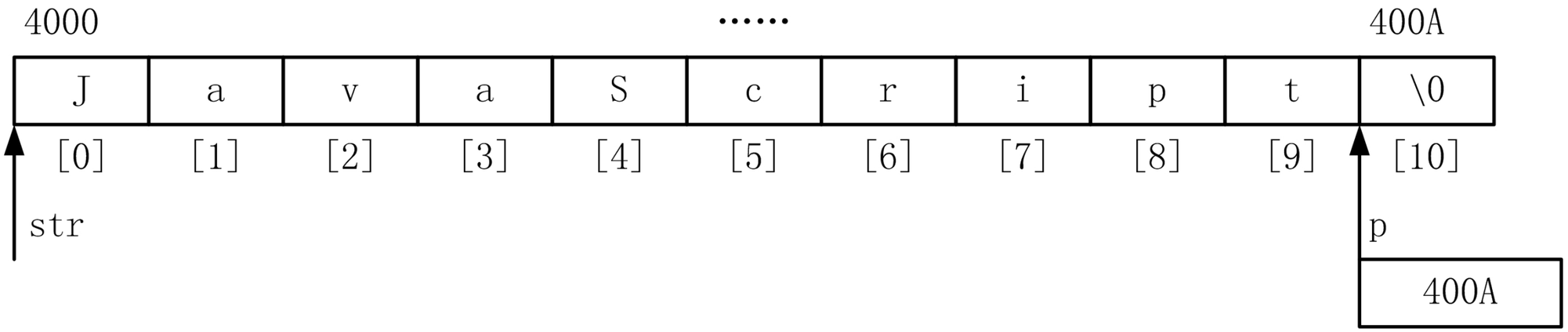
程序运行屏幕

```
strlen=10
```

JavaScript↙

7.4.1 指向字符串的指针

图7.20 指针相减的含义



当p指向空字符时，转向第7行，如图所示。p-str的结果是两个地址间字符元素的数目，正好是字符串的长度（不计空字符）。

7.4.1 指向字符串的指针

例7.15

```
1 #include <stdio.h>
2 int main()
3 {
4   char str[100],*p=str;
5   scanf("%s", str); //输入字符串
6   while (*p) p++; //指针p指向到字符串结束符
7   printf("strlen=%d\n",p-str); //输出字符串长度
8   return 0;
9 }
```

这个例子中，可不可以将str定义为“char *str”？答案是否定的。程序输入字符串，即多个字符元素，能存储它的数据类型只能是字符数组。而字符指针只是指向字符串，并不能实际存储字符串。

7.4.1 指向字符串的指针

例7.15

```
1 #include <stdio.h>
2 int main()
3 {
4    char str[100],*p=str;
5    scanf("%s", str); //输入字符串
6    while (*p) p++; //指针p指向到字符串结束符
7    printf("strlen=%d\n",p-str); //输出字符串长度
8    return 0;
9 }
```

一旦将str定义为“char *str”，第5行就是使用了无效地址。即使str指向有效地址，其存储空间也没有足够的长度来接受输入，第5行仍然会使存储空间越界而导致严重错误。

7.4.1 指向字符串的指针

- 请记住，指针可以指向数组，使得数组的访问多了一种途径，但指针并不能替代数组来存储大批量数据。

7.4.2 指针与字符数组的比较

- 由于数组和指针之间的密切关系，用字符数组和字符指针变量都能实现字符串的表示和运算。例如：

```
char s[100]="Computer";
char *p="Computer";
```

- 特别的，任何传递字符数组或字符指针的函数都接受两种方式的参数，但它们二者之间是有显著差异的。

7.4.2 指针与字符数组的比较

- 1．存储内容不同
- 字符数组能够存放字符串所有字符和结束符，字符指针仅存放字符串的首地址。即定义字符数组，系统会为其分配指定长度的内存单元，而定义指针变量，系统只分配4个字节的内存单元用于存放地址。

7.4.2 指针与字符数组的比较

- 2．运算方式不同
- 字符数组s和字符指针p尽管都是字符串的首地址，但s是数组名，是一个指针常量，不允许做左值和自增自减运算。而p是一个指针变量，允许做左值和自增自减运算。
- 作为地址值，s在程序运行期间不会发生变化，而p是可变的。

7.4.2 指针与字符数组的比较

- 3．赋值操作不同
- 字符数组s可以进行初始化，但不能使用赋值语句进行整体赋值，只可以按元素来赋值。例如：

```
s="C++"; //错误
s++; //错误
s[0]='C'; //正确
```

7.4.2 指针与字符数组的比较

- 字符指针既可以进行初始化，也可以使用赋值语句。例如：

```
p="C++"; //正确
*p='C'; //正确
p++; //正确
```

7.4.2 指针与字符数组的比较

- 一般地，如果程序中需要可以变化的字符串，则要建立一个字符数组，通过一个指向它的字符指针变量来访问其中的字符元素。例如：

```
char str[100], *pz=str;
```

7.4.2 指针与字符数组的比较

【例7.16】

使用字符指针下标法访问字符串。

7.4.2 指针与字符数组的比较

例7.16

```
1 #include <stdio.h>
2 int main()
3 {
4   char *p="VisualBasic";
5   int i=0;
6   while (p[i]) printf("%c",p[i++]); //输出结束符之前的全部字符
7   return 0;
8 }
```

程序运行结果如下：

VisualBasic

7.4.3 指向字符串数组的指针

- 字符串数组是一个二维字符数组，例如：

```
char sa[6][7]={"C++","Java","C","PHP","CSharp","Basic"};
```

按一维数组的角度来看，数组sa有6个元素，每个元素均是一个一维字符数组，即字符串。因此数组sa可以理解为包含6个字符串的一维数组，例子中的初值正是字符串的写法。如图所示。

7.4.3 指向字符串数组的指针

图7.21 字符串数组的内存形式

sa

sa[0]	C	+	+	\0			
sa[1]	J	a	v	a	\0		
sa[2]	C	\0					
sa[3]	P	H	P	\0			
sa[4]	C	S	h	a	r	p	\0
sa[5]	B	a	s	i	c	\0	

7.4.3 指向字符串数组的指针

- 由于一个字符指针可以指向一个字符串，为了用指针表示字符串数组，需要使用指针数组。例如：

```
char *pa[6]={"C++","Java","C","PHP","CSharp","Basic"};
```

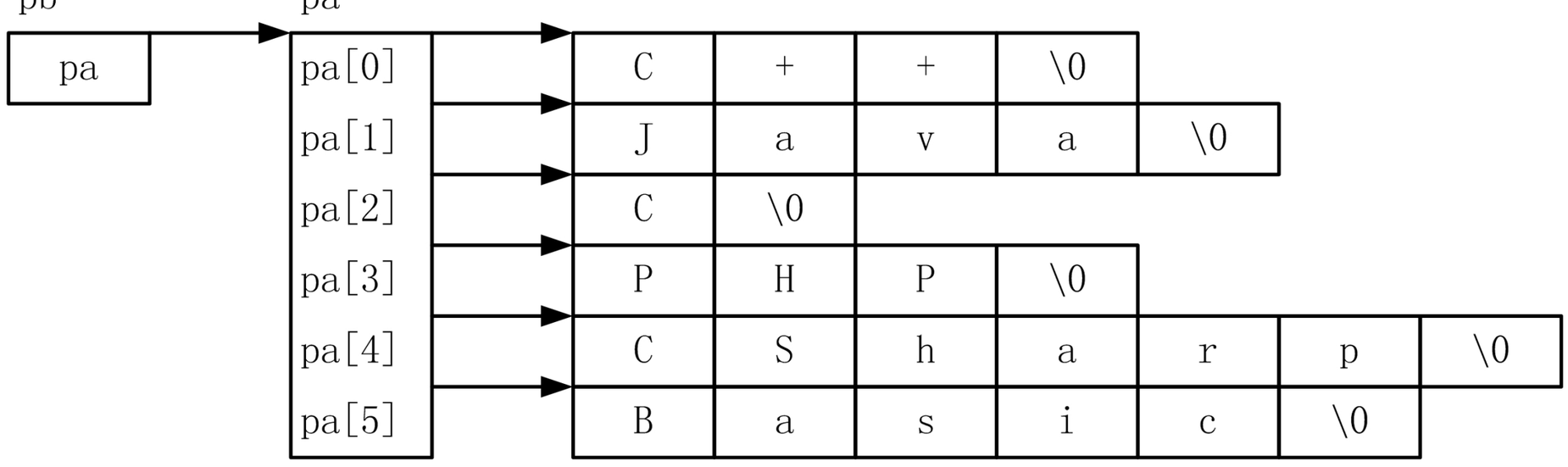
- 其中pa为一维数组，有6个元素，每个元素均是一个字符指针。

7.4.3 指向字符串数组的指针

- 为了通过指针方式使用指针数组pa，还可以定义指向指针的指针。例如：

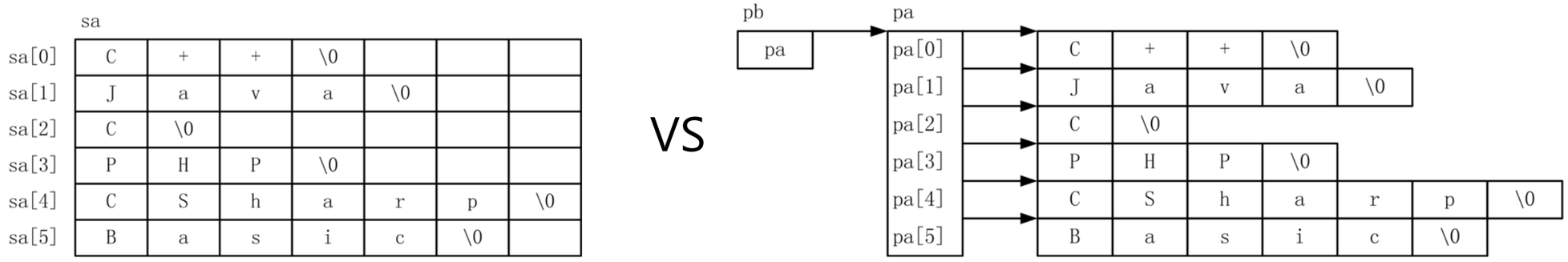
```
char **pb=pa;
```

图7.22 指向字符串数组的指针



7.4.3 指向字符串数组的指针

- 用字符串数组存储若干个字符串时，由于二维数组每一行包含的元素个数要求相等，因此需要取最大的字符串长度作为列数。而实际应用中的各个字符串长度一般是不相等的，若按最长字符串来定义列数，必然会浪费内存单元。
- 若使用字符指针数组，各个字符串按实际长度存储，指针数组元素只是各个字符串的首地址，不存在浪费内存单元问题



7.4.3 指向字符串数组的指针

- 在计算机信息处理中，对字符串的操作是最常见的，如果使用指针方式，会大大提高处理效率。例如，对若干字符串使用冒泡法按字母排序，如果用数组方式，比较交换时会产生字符串复制的开销。若使用字符指针数组，只需交换指针值改变指向，而字符串本身无需做任何操作。

7.4.3 指向字符串数组的指针

【例7.17】

将若干字符串使用冒泡法由小到大排序。

7.4.3 指向字符串数组的指针

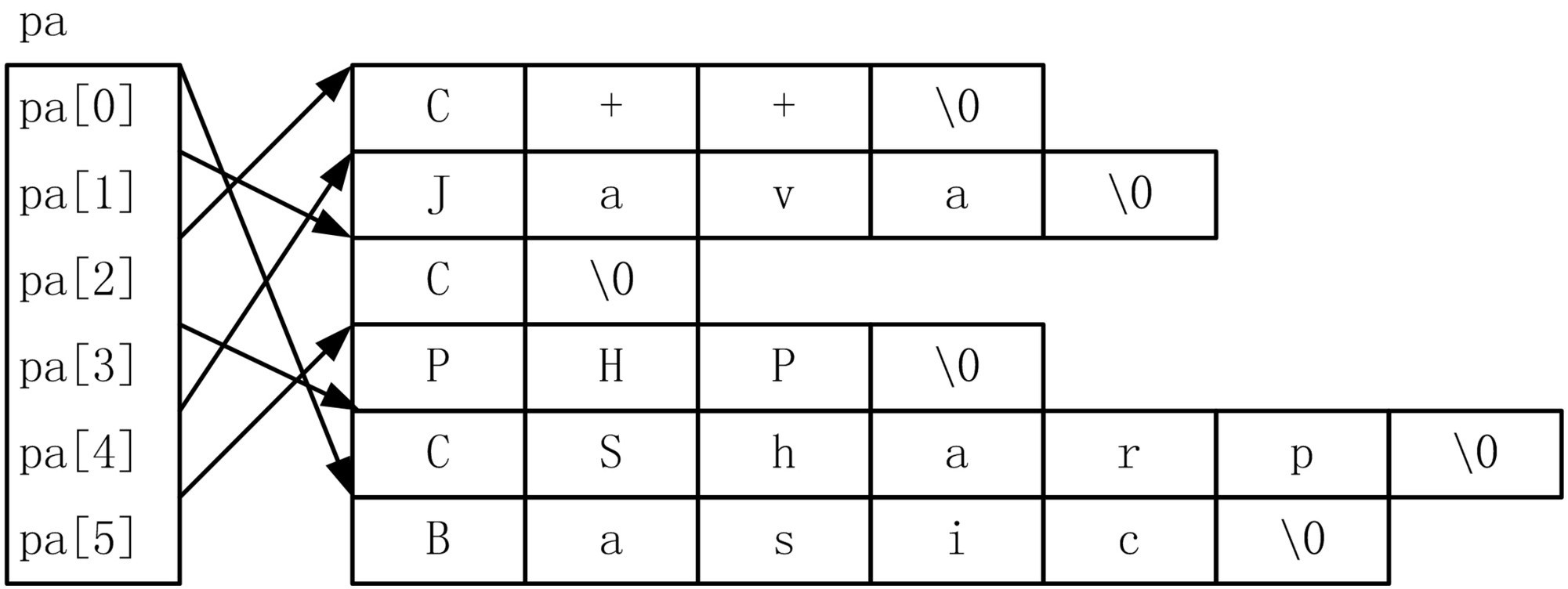
例7.17

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 int main()
 4 {
 5   char*pa[6]={"C++","Java","C","PHP","CSharp","Basic"},*t;
 6   int i, j;
 7   for(j=0; j<6-1; j++)
 8     for(i=0; i<6-1-j; i++)
 9       if(strcmp(pa[i],pa[i+1])>0)
10         t=pa[i] , pa[i]=pa[i+1] , pa[i+1]=t; //指针交换
11   for (i=0; i<6; i++) printf("%s ",pa[i]);
12   return 0;
13 }
```

7.4.3 指向字符串数组的指针

排序后如图所示。

图7.23 利用指针交换进行排序




程序设计