



C语言程序设计基础

林川

第8章 指针



- 变量、内存单元、地址
- 指针类型
 - 定义、初始化、赋值、使用
- 指针应用
 - 数据传递
 - 字符串处理
 - 内存分配
 - 数据遍历



8.1.2 指针类型

- 指针是一种新的数据类型
 - 存放变量的地址
 - 存放数据单元的地址

```
int x, *p;
```

那么可将变量x的地址存在指针p中：

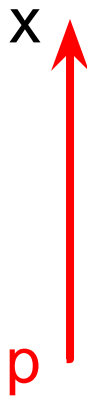
```
p = &x;
```

& 是取地址运算符 scanf("%d", &x)

变量和数据单元地址

地址	内存单元

300	100
304	1
308	155
.....
600	300



```
int x = 100;  
int *p;  
p = &x;
```

表达式 $*p$ 和 变量 x
指代同一个东西

$p = \&x$



8.1.3 指针变量的定义

类型名 * 指针变量名

int *p;

- p 是整型指针, 可用来指向整型变量
- 只能指向同类型的变量

float *fp;

- fp 是浮点型指针, 可用来指向浮点型变量

double *q;

- q 是字符型指针, 可用来指向double变量

char *cp;

- cp 是字符型指针, 可用来指向字符型变量



8.1.4 指针的基本运算

- 给指针赋值

```
int a, x, *p;
```

```
p = &a;
```

- 访问指针所指向的变量

```
*p = 3;   a = 3
```

```
*p = 5;   a = 5
```

```
x = *p;   x = 5
```



[例8-2]指针运算和访问

```
int a = 3, *p;  
p = &a;  
printf("a=%d, *p=%d\n", a, *p); // a = 3, *p = 3  
*p = 10;  
printf("a=%d, *p=%d\n", a, *p); // a = 10, *p = 10  
printf("Enter a: ");           /*若输入5*/  
scanf("%d", &a);               a = 5  
printf("a=%d, *p=%d\n", a, *p); // a = 5, *p = 5  
(*p)++;  
printf("a=%d, *p=%d\n", a, *p); // a = 6, *p = 6
```



8.1.5 指针的变量的初始化

```
int a, x;
```

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

```
int * q = NULL;
```

NULL是一个常量，值为0，表示空指针

```
#define NULL 0
```

```
int * q = 0;
```

```
float * fp = (float*)1732;
```




8.2 变量交换swap函数实现

```
void swap1(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

```
void main()
{
    int a=1, b=2;
    swap1(a,b);
}
```

能否成功交换变量
a和b的值？

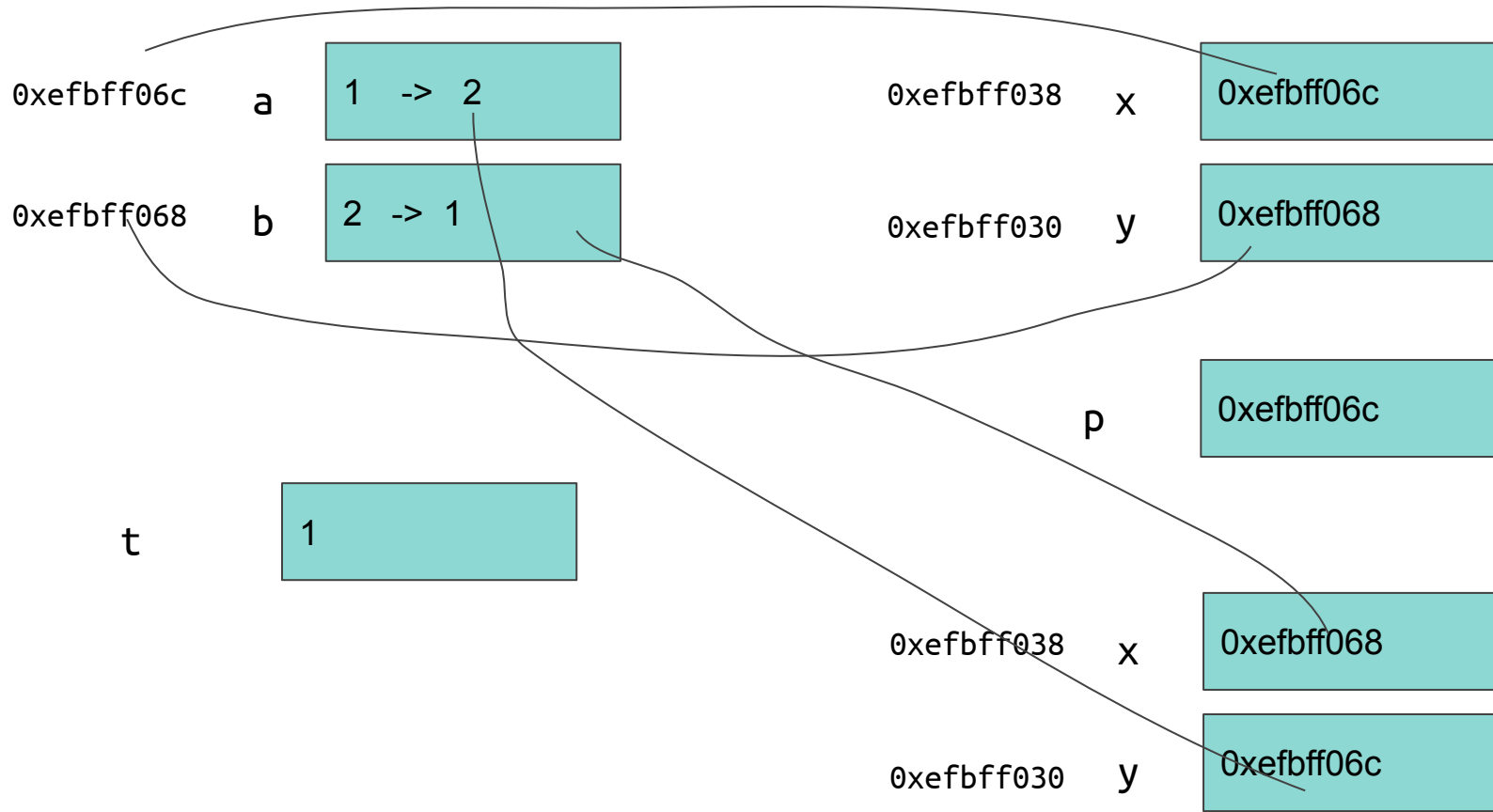


8.2 变量交换swap函数实现

```
void swap2(int *x, int *y)
{
    int * p = x;
    x = y;
    y = p;
}
```

```
void main()
{
    int a=1, b=2;
    swap2(&a, &b);
}
```

能否成功交换变量a和b的值？





8.2 变量交换swap函数实现

```
void swap3(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```
void main()
{
    int a=1, b=2;
    swap3(&a, &b);
}
```

能否成功交换变量a和b的值？



8.2.2 指针作为函数的参数

```
void swap3(int *x, int *y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}

void main()
{
    int a=1, b=2;
    swap3(&a, &b);
}
```

传递结果
改变主调函数的变量值

例[8-4]编写函数，计算某年某天对应的月份和日期



函数的输入参数：年份，天数

```
int year, int yearday
```

例如2008年的第128天, year=2008, yearday=128

函数的输出结果：月份，日期

结果不止一个，无法用返回值

可以用指针变量

```
int *pmonth, int *pday
```

函数原型：

```
void month_day(int year, int yearday,  
               int *pmonth, int *pday);
```



例[8-4]计算某年某天对应的**月份**和**日期**

```
void month_day (  
    int year,  int yearday,  
    int *pmonth, int *pday)  
{  
    int k, leap;  
    int tab [2][13] = {  
        {0,31,28,... 31 },  
        {0,31,29,... 31 }};  
  
    int year = 2020; int yearday = 200;  
    int month, day;  
  
    month_day(year, yearday, &moth, &day);
```

```
/* 闰年判别leap */  
leap = (year%4==0 &&  
        year%100!=0) ||  
        year%400== 0;  
  
for( k=1;  
    yearday > tab[leap][k];  
    k++ )  
    yearday -= tab[leap][k];  
  
*pmonth = k;  
*pday = yearday;  
}
```



8.3 指针与数组

数组名实际上代表了一个指针

- 它指向数组的首元素

```
int a[100];
```

那么a就是一个指针，存储了a[0]的地址。

数组名是一个指针常量

- 不能改变它的地址值

```
int a[100], c, *p;
```

```
a = & c; /*不可以*/
```

```
p = a; /*可以*/
```




8.3 指针与数组

```
short a[100];
```

```
short *p = a;
```

指针p的地址值为3000

假设短整数为2个字节长

指针p+1的地址值为3002

...

指针p+i的地址值为3000+2i

$$p + i == a + i$$

(都指向a的第i个元素)

指针的加法：指针 + 整数n

结果：将指针往后移动n个单元

即：地址值增加了 $n * \text{sizeof}(\text{数据类型})$

内存地址

内存单元

3000

a[0]

3002

a[1]

...

...

...

...

3000+2i

a[i]

...

...

...

...

3198

a[99]

...

...



指针比较与减法

两指针可以比较大小

其结果等价于比较它们地址的大小

两同类型指针可相减

其结果等于它们之间所能存储的数据个数

即：
$$\frac{p\text{地址} - q\text{地址}}{\text{sizeof(类型)}}$$

$$p - q = \frac{\text{p地址} - \text{q地址}}{\text{sizeof(类型)}}$$

`p = a; q = &a[4];`

`q - p` 等于多少?

4

`(int)q - (int)p` 等于多少?

32

8.3 指针与数组



```
int a[100];
```

指针a指向首元素a[0];

指针a+i指向元素a[i];

表达式a[i]等价于

$*(a+i)$

内存地址

内存单元

3000

a[0]

3002

a[1]

...

...

...

...

3000+2i

a[i]

...

...

...

...

3198

a[99]

...

...



例[8-5]冒泡排序

把最大值放到数据的最后

对剩下的数据进行重复处理

-- 流程整体上和选择法排序一样

-- 区别在于如何把最大值放到最后



例[8-5]冒泡排序

```
void bubble(int a[], int n)
/* void bubble(int *a, int n) */
{
    int i, j, t;
    for(i=1; i<n; i++)
        for(j=0; j<n-i; j++)
            if(a[j]>a[j+1])
            {
                t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
}
```

在 $a[0], a[1], \dots, a[j]$ 中 $a[j]$ 最大

交换, 使得 $a[j+1]$ 在 $a[0], a[1], \dots, a[j+1]$ 中最大



例[8-5]冒泡排序

```
void main()
{
    int n, a[8];
    /* 输入 n 和 n个值到数组a中 */
    .....
    bubble(a,n);
    /* 输出排序后的数组a */
    .....
}
```

将数组a的首地址作为参数
传给函数bubble



8.3.3 数组名作为函数参数

```
int sum(int a[], int n) // int sum(int *a, int n)
/*int a[]是形式参数int *a的等价写法*/
{
    int i, s;
    for( s = 0, i=0; i<n; i++ )
        s += a[i];
    return s;
}
```



8.3.3 数组名作为函数参数

```
int sum(int a[], int n) /*int a[]等价int *a*/  
{  
    int i, s;  
    for( s = 0, i=0; i<n; i++ )  
        s += a[i];  
    return s;  
}
```

假设有int b[100]; 那么：

sum(b, 100)的结果是 b[0]+b[1]+...+b[99]

sum(b, 88) 的结果是 b[0]+b[1]+...+b[87]

sum(b+7,9) 的结果是 b[7]+b[8]+...+b[15]

sum(&b[7],9) 的结果也是 b[7]+b[8]+...+b[15]



例8-7编写函数将数组逆序

```
void reverse(int a[], int n)
{
    int i,k,t;
    for( i=0, k=n-1; i<k; i++,k--)
        t = a[i], a[i] = a[k], a[k] = t;
}

void main()
{
    int a[10], k;
    for( k=0; k<10; k++ )    a[k] = k;
    reverse(a+1, 9);
    for( k=0; k<10; k++ )    printf("%d ", a[k]);
}
```

输出为 : 0 9 8 7 6 5 4 3 2 1



8.4 简单的加密问题

```
void encrypt(char *s)
{
    for( ; *s; s++ )
        *s = ( (*s=='z') ? 'a' : *s+1);
}

void main ()
{
    char pwd[100];
    gets(pwd);
    encrypt(pwd);
    printf(pwd);
}
```

如果输入为 : Hello Hangzhou
那么输出为 : Ifmmp!Ibohaipv

- 字符串结束标志0
- 字符数组
- 字符指针



力扣提交通过：

<https://leetcode-cn.com/problems/two-sum/submissions/>

<https://leetcode-cn.com/problems/how-many-numbers-are-smaller-than-the-current-number/>

描述清楚解题步骤(流程)

<https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/>



8.4.2 字符串和字符指针

字符串常量

"array"

"point"

- 用一对双引号括起来的字符序列
- 被看做一个特殊的一维字符数组,在内存中连续存放
- 实质上是一个指向该字符串首字符的指针常量

```
char *sp = "point";
```



8.4.2 字符串和字符指针

- 字符串常量

```
char sa[] = "array\0";
```

```
char *sp= "point\0";
```

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

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

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

```
printf("%c ", "string"[5]);
```

输出为: array point string g

```
printf("%s ", sa+2);
```

```
printf("%s ", sp+3);
```

```
printf("%s ", "string"+1);
```

输出为:

ray nt ring

字符串常量是表达式
其值为一个指针, 指向存储它的地址



8.4.2 字符串和字符指针

```
char sa[] = "array";
```

定义了一个数组, 并被初始化为

array\0

```
char *sp= "point";
```

定义了一个指针sp, 而非数组。指针指向了一个字符串常量, 而 该字符串常量存储在何处呢?

Somewhere decided by the system.



8.4.2 字符串和字符指针

字符串输出

```
char *p;  
puts(p);  
printf("%s", p);
```

要求指针p指向了一个以\0结尾的字符串

字符串输入

```
gets(p);  
scanf("%s", s);
```

要求指针p所指的地方具有足够的free存储空间。否则，程序崩溃



8.4.3 常用的字符串处理函数

- gets/scanf
 - gets函数读入一整行(包括空格), 直到回车为止
 - scanf函数读入字符, 直到空格和回车为止
- puts/printf
 - puts函数输出后会 **自动换行**



8.4.3 常用的字符串处理函数

- strcpy(char *s, char *t)
字符串复制: $s = t$
- strcat(char *s, char *t)
字符串连接: $s += t$
- strcmp(char *s, char *t)
字符串比较: $s - t$
 - strcmp("abc", "aba")的结果 **大于** 0
 - strcmp("abc", "abcd")的结果 **小于** 0
 - strcmp("abc", "abc")的结果 **等于** 0



8.4.3 常用的字符串处理函数

- `strlen(char *s)`

字符串的长度(不包括\0)

`strlen("abc", "aba")`的结果是3

`#include <string.h>`

或者

`#include <stdlib.h>`



8.4.3 常用字符串处理函数实现参考

```
char* strcpy(char *s, char *t)
```

```
{  
    do {  
        *s++ = *t;  
    } while(*t++);  
    return s;  
}
```

8.4.3 常用字符串处理函数实现参考



```
char* strcat(char *s, char *t)
{
    while ( *s )
        s++;
    strcpy(s,t);
    return s;
}
```



8.4.3 常用字符串处理函数实现参考

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



8.4.3 常用字符串处理函数实现参考

```
char* strcmp(char *s, char *t)
```

```
{  
    while( *s==*t && *s )  
        s++, t++;  
    return (*s - *t);  
}
```



8.4.3 常用字符串处理函数实现参考

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



8.4.3 常用字符串处理函数

- 假设`strlen(s)`等于20, 那么`strlen(s+5)`等于几?

15

- 如果有字符串`s`为“hello”, 字符串`t`为“world\0”, 那么`strcat(s,t)`是?

helloworld

`strcat(s+1,t+1)`是?

elloorld



8.4.3 常用字符串处理函数

- 执行strcpy(s, "good\0morning")后, 字符指针s所指向的字符串为

"good"

- 下列语句能够正确执行吗？

```
strcpy("old string","new");
```

不能改变字符串常量



8.5 动态内存申请和使用

- 数组的局限
 - 大小固定、使用不灵活
 - 预定义的数组容量通常比较小
- 动态分配内存
 - 根据运行情况, 按需分配



[例8-12] 数据输入和求和

- 输入一个整数 n , 以及 n 个整数, 计算它们的和。

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

```
#include<stdlib.h>
```

```
int n, sum, i, *p;
```

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

```
p = (int*) calloc(n, sizeof(int));
```



[例8-12] 数据输入和求和

```
/* 检查内存申请结果 */  
if( p==NULL ) /* 等价于 if ( !p ) */  
{  
    /* 输出一些信息, 作为错误提示 */  
    printf("Not able to allocate memory\n");  
    return;  
}
```



[例8-12] 数据输入和求和

```
/* 输入n个整数 */  
for( i=0; i<n; i++ )  
    scanf("%d",p+i);    /* &p[i] */  
  
/* 求和 */  
for(sum=0, i=0; i<n; i++ )  
    sum += p[i]; /* *(p+i) */  
printf("sum=%d\n", sum);
```

[例8-12] 数据输入和求和



/* 释放内存 */

```
free( p );
```



8.5.2 用指针实现内存动态分配

- 包含头文件
 - `#include <stdlib.h>`
 - `#include <malloc.h>`



8.5.2 用指针实现内存动态分配

- 调用内存分配函数

- `void * calloc(unsigned n, unsigned size)`

- 参数 n – 包含的元素个数

- 参数 size – 每个元素所占的字节数

- (总的内存大小为 $n \times \text{size}$ 字节)

- 返回值 – 无类型指针

`void` 无类型

`void *` 无类型的指针

`void *` 可以强制转化为任何类型的指针



8.5.2 用指针实现内存动态分配

- `void * malloc(unsigned size)`
参数 `size` – 申请内存的字节数
(总的内存大小为 `size` 字节)
返回值 – 无类型指针
指向所分配的内存地址

假设你的数据类型为 `UType`, 那么调用步骤如下 :

```
UType *p;
```

```
p = (UType *) malloc( 元素个数*sizeof( UType) );
```

```
if( !p ) /*检查分配是否成功*/
```

```
..... /* 处理错误 */
```



8.5.2 用指针实现内存动态分配

- 然后就可以如普通数组一样使用p
- 使用完成之后, 释放内存

`free(p);`



8.5.2 用指针实现内存动态分配

- 使用过程中如果发现内存不够多(或者多了), 还可以动态调整

`void * realloc(void *p, unsigned size)`

指针p 必须是指向动态申请的内存, 否则出错
size为新的大小

- 如果调整成功, 返回新的地址, 并将原地址的内容复制到新地方
- 否则, 返回NULL。



8.5.2 内存动态调整

```
int count = 0;  
float * pbuf = NULL;  
int bufsize = 0, delta = 1000;
```

```
int checkbuf()  
{  
    void * p = NULL;  
    if( count < bufsize ) return 1;  
    p = realloc( pbuf, (bufsize+delta)*sizeof(float) );  
    if( ! p ) return 0;  
    pbuf = (float*) p;  
    bufsize += delta;  
    return 1;  
}
```



8.5.2 内存动态调整

```
int addToBuf(float v)
{
    if( !checkbuf() ) return 0;
    pbuf[count++] = v;
    return count;
}
```

本章要点



- 变量、内存单元和地址之间是什么关系？
- 如何定义指针变量, 怎样才能使用指针变量？
- 什么是指针变量的初始化？
- 指针变量的基本运算有哪些？如何使用指针操作所指向的变量？
- 指针作为函数参数的作用是什么？
- 如何使用指针实现函数调用返回多个值？
- 如何利用指针实现内存的动态分配？