指针进阶

1 字符指针

字符指针的用法：（1）指向字符变量。（2）指向字符串的首地址。如下：

#include<stdio.h>

int main()

{

char ch = 'q';

char\* p = &ch;//指向字符变量

char\* p1 = "hello!!";//指向字符串首地址

printf("%c\n", \*p);

printf("%c\n", \*p1);

return 0;

}

例：读程序

#include<stdio.h>

int main()

{

char str1[] = "hello!!";//创建两个数组，肯定存放在不同的空间

char str2[] = "hello!!";

const char\* str3 = "hello!!";//由于赋值右边是常量字符串，其值不能被修改，所以只需要一个空间即可

const char\* str4 = "hello!!";//由此两个字符指针都指向常量字符串的首地址，是相同的

if (str1 == str2)//代表的是数组首地址，不同的

printf("str1 and str2 are same!\n");

else

printf("str1 and str2 are not same!\n");

if (str3 == str4)//指针代表的是地址，由于指向同一地方，所以相同

printf("str3 and str4 are same!\n");

else

printf("str3 and str4 are not same!\n");

return 0;

}

2 指针数组

指针数组，本质上是数组，数组中存放的是指针（地址）。

#include<stdio.h>

int main()

{

int a[] = { 1,2,3,4,5 };

int b[] = { 2,3,4,5,6 };

int c[] = { 3,4,5,6,7 };

int\* arr[] = { a,b,c };//利用指针来指向数组首地址，那么这个指针数组表示的内容更多

int i = 0;

int j = 0;

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

{

for (j = 0; j < 5; j++)

{

printf("%d ", \*(arr[i] + j));

}

printf("\n");

}

return 0;

}

3 数组指针

数组指针的认识：

#define \_CRT\_SECURE\_NO\_WARNINGS 1

//数组指针本质上是指针，例如整型指针表示指向整型的指针，则数组指针表示为指向数组的指针。

#include<stdio.h>

int main()

{

int arr[10] = { 0 };

//arr-数组名代表的是数组首元素地址，即arr[0]的地址，而并非数组的地址

int(\*p2)[10] = &arr;//数组指针，指向数组地址的指针，&arr才为数组的地址

//数组指针的定义，括起来表示指针 指向十个元素的地址 元素的类型

double\* brr[5] = { 0 };

double\* (\*pb)[5] = &brr;

//arr和&arr是值相同但意义完全不同的两个变量

int\* p1 = arr;

printf("%p\n", p1);//p1是整型指针，+1跳过一个整型地址

printf("%p\n", p1 + 1);

printf("%p\n", p2);

printf("%p\n", p2 + 1);//p2是数组指针，+1跳过一整个数组的地址

return 0;

}

数组指针的应用：

#include<stdio.h>

void print1(int arr[][5], int r, int c)//数组必须对应

{

int i = 0;

int j = 0;

for (i = 0; i < r; i++)

{

for (j = 0; j < c; j++)

{

printf("%d ", arr[i][j]);

}

printf("\n");

}

}

void print2(int (\*pa)[5], int r, int c)//这里pa代表的是第一行元素的地址，是数组指针

{

int i = 0;

int j = 0;

for (i = 0; i < r; i++)

{

for (j = 0; j < c; j++)

{

printf("%d ", \*((\*(pa+i))+j));//\*pa表示的是某行元素的首地址 pa+i表示进入下一行元素的打印

}

printf("\n");

}

}

int main()

{

int arr[3][5] = { {1,2,3,4,5},{2,3,4,5,6},{3,4,5,6,7} };

print1(arr, 3, 5);

print2(arr, 3, 5);//二维数组的数组名表示第一行元素的地址，即一维数组的地址

return 0;

}

例：各定义的含义

#include<stdio.h>

int main()

{

int prr1[5];//prr1是一个整型数组

int\* prr2[5];//prr2是一个指针数组，包含五个指针

int(\*prr3)[5];//prr3是一个数组指针，指针指向一个含有五个元素的数组

int(\*prr4[10])[5];//prr4是一个拥有十个指针的数组，且每个指针指向一个含有五个元素的数组，且数组元素为整型

return 0;

}

4 数组传参和指针传参

一维数组传参：

#include<stdio.h>

void test0(int arr[])//数组传参数组接收

{}

void test0(int arr[10])

{}

void test0(int\* arr)//数组名是首元素地址，指针接收

{}

void test1(int\* arr[])//数组传参，数组接收，没有问题

{}

void test1(int\* arr[20])

{}

void test1(int\*\* arr[])//指针数组的数组名是首元素地址，且首元素也是指针，即二级指针

{}

int main()

{

int arr0[10] = { 0 };

int\* arr1[20] = { 0 };

test0(arr0);//一维数组传参

test1(arr1);//指针数组传参

return 0;

}

二维数组传参：

**数组方式**：形参行可以省略，但列不可以省略。

**指针方式**：二维数组的数组名是第一行数组的地址，即数组的地址，必须有数组指针来作为形参。

**指针初阶练习讲解：**

（1）读程序

#include<stdio.h>

int i;//i是全局变量，没有赋初值，默认为0

int main()

{

i--;//i=-1

if (i > sizeof(i))//sizeof操作符返回的结果是unsigned int 类型，当比较时会把i也转换成unsigned int 类型处理，则i会变成一个很大的数

printf(">\n");//

else

printf("<\n");

return 0;

}

（2）写代码：Sn=a+aa+aaa+aaaa+…如2+22+222+2222+22222+…

#include<stdio.h>

int main()

{

int i = 0;

int a = 0;

int n = 0;

int sum = 0;

int ret = 0;

scanf("%d", &a);

scanf("%d", &n);

for (i = 0; i < 5; i++)

{

ret = ret \* 10 + a;//后一项与前一项的关系

sum += ret;

}

printf("sum = %d\n", sum);

return 0;

}

（3）写代码：求出1-100000中的水仙花数，水仙花数表示数的每一位的位数次方之和为其本身的数，如153 = 3^3+5^3+1^3，则153是一个水仙花数。

一、繁琐且不严谨的方式：

#include<stdio.h>

#include<math.h>

int main()

{

int i = 0;

int j = 0;

int n = 0;

int b[10] = { 0 };

for (i = 0; i <= 100000; i++)

{

if (i >= 0 && i < 10)

{

n = 1;

if (pow(i,n) == i)

printf("%d ", i);

}

if (i >= 10 && i < 100)

{

n = 2;

b[0] = i % 10;

b[1] = i / 10;

if(pow(b[0],n) + pow(b[1], n) == i)

printf("%d ", i);

}

if (i >= 100 && i < 1000)

{

n = 3;

b[0] = i % 100 %10;

b[1] = i % 100 / 10;

b[2] = i / 100;

if (pow(b[0], n) + pow(b[1], n) + pow(b[2], n) == i)

printf("%d ", i);

}

if (i >= 1000 && i < 10000)

{

n = 4;

b[0] = i % 1000 % 100 % 10;

b[1] = i % 1000 % 100 / 10;

b[2] = i % 1000 /100;

b[3] = i / 1000;

if (pow(b[0], n) + pow(b[1], n) + pow(b[2], n) + pow(b[3], n) == i)

printf("%d ", i);

}

if (i >= 10000 && i < 100000)

{

n = 5;

b[0] = i % 10000 % 1000 % 100 % 10;

b[1] = i % 10000 % 1000 % 100 / 10;

b[2] = i % 10000 % 1000 / 100;

b[3] = i % 10000 / 1000;

b[4] = i / 10000;

if (pow(b[0], n) + pow(b[1], n) + pow(b[2], n) + pow(b[3], n) + pow(b[4], n) == i)

printf("%d ", i);

}

if (i == 100000)

{

n = 6;

if (pow(1, n) == i)

printf("%d ", i);

}

}

printf("\n");

return 0;

}

二、简洁的方式：写代码要讲顺序和逻辑

#include<stdio.h>

#include<math.h>

int main()

{

int i = 0;

for (i = 0; i <= 100000; i++)

{

//1、计算i的位数

int n = 1;

int tmp = i;

while (tmp / 10)

{

n++;

tmp = tmp / 10;

}

//2、计算i的每一位的位数次方和

double sum = 0;

tmp = i;

while (tmp)

{

sum += pow(tmp % 10, n);

tmp = tmp / 10;

}

//3、判断

if (sum == i)

{

printf("%d ", i);

}

}

return 0;

}

（4）指针-指针得到的是指针之间元素的个数，而并非字节数。

（5）打印菱形的\*号，通用性强。

#include<stdio.h>

int main()

{

int i = 0;

int j = 0;

int line = 0;

scanf("%d", &line);

//1.上半部分

for (i = 0; i < line; i++)

{

//（1）打印每一行

//打印空格

for (j = 0; j < line - 1 - i; j++)

{

printf(" ");

}

//打印\*号

for (j = 0; j < 2 \* i + 1; j++)

{

printf("\*");

}

printf("\n");

}

//2.下半部分

for (i = 0; i < line - 1; i++)

{

//（1）打印每一行

//打印空格

for (j = 0; j <= i; j++)

{

printf(" ");

}

//打印\*号

for (j = 0; j < 2 \* (line - 1 - i) - 1; j++)

{

printf("\*");

}

printf("\n");

}

return 0;

}

（5）喝汽水，一元一瓶汽水，两个空瓶可以换一瓶汽水，问20元总共可以喝多少瓶汽水？

#include<stdio.h>

int main()

{

int money = 0;

int sum = 0;

int empty = 0;//真实的空瓶数量

scanf("%d", &money);

sum = money;//最初可以喝的汽水

empty = money;

while (empty >= 2)//换的汽水继续换购

{

sum += empty / 2;//喝完之后换购的汽水

empty = empty / 2 + empty % 2;//真实的空瓶数量

}

printf("sum = %d\n", sum);

return 0;

}

（6）写代码：一个整数数组，将其中的奇数放在数组的前半部分，偶数放在后半部分。

#include<stdio.h>

void swap(int arr[], int n)

{

int i = 0;

int j = 0;

int tmp = 0;

//类似冒泡排序法，每次相邻两个数进行判断

for (i = 0; i < n-1; i++)//跑n-1趟

{

for (j = i; j < n - 1; j++)//每趟比较n - i -1次

{

if (arr[j] % 2 == 0 && arr[j + 1] % 2 == 1)

{

tmp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = tmp;

}

}

}

}

int main()

{

int n = 0;

int i = 0;

int arr[20] = { 0 };

scanf("%d", &n);

for (i = 0; i < n; i++)

{

scanf("%d", &arr[i]);

}

swap(arr, n);

for (i = 0; i < n; i++)

{

printf("%d ", arr[i]);

}

return 0;

}

（7）读程序，常数是整型，截断，运算和打印都存在整型提升，注意补码原码和反码。

（8）打印一个简单的杨辉三角。

#include<stdio.h>

int main()

{

int i = 0;

int j = 0;

int n = 0;

int arr[20][20] = { 0 };//二维数组存储数据

scanf("%d", &n);

arr[0][0] = 1;

arr[1][0] = 1;

arr[1][1] = 1;//前两行

for (i = 2; i < n; i++)//第三行开始

{

//每一行，首1，中间数，尾1

arr[i][0] = 1;

for (j = 1; j < i; j++)

{

arr[i][j] = arr[i - 1][j - 1] + arr[i - 1][j];

}

arr[i][i] = 1;

}

for (i = 0; i < n; i++)

{

for (j = 0; j < n; j++)

{

if(arr[i][j])

{

printf("%d ", arr[i][j]);

}

}

printf("\n");

}

return 0;

}

（9）猜凶手。四个嫌疑人，A：不是我；B：是C；C：是D；D：C在胡说。假设四个人中有三个人说的真话，一个人说的假话，则判断谁是凶手。（智力题发散思维）牛客网-智力题

方法一：

#include<stdio.h>

int main()

{

int a = 0;

int b = 0;

int c = 0;

int d = 0;

a = 1;//假设凶手

if (a)

{

//判断是否说谎 0 假话；1 真话

a = 0;

b = 0;

c = 0;

d = 1;

if (a + b + c + d == 3)//说真话的人数为3，则假设成立，找到凶手。

printf("凶手是a\n");

}

b = 1;

if (b)

{

a = 1;

b = 0;

c = 0;

d = 1;

if (a + b + c + d == 3)

printf("凶手是b\n");

}

c = 1;

if (c)

{

a = 1;

b = 1;

c = 0;

d = 1;

if (a + b + c + d == 3)

printf("凶手是c\n");

}

d = 1;

if (d)

{

a = 1;

b = 0;

c = 1;

d = 0;

if (a + b + c + d == 3)

printf("凶手是d\n");

}

return 0;

}

方法二：

#include<stdio.h>

int main()

{

char killer = 0;

for (killer = 'A'; killer <= 'D'; killer++)//循环假设凶手

{

if ((killer != 'A') + (killer == 'C') + (killer == 'D')+(killer != 'D') == 3)//判断四个人说的真话还是假话

{//如果真话的数量==3，则此时的killer即为凶手。

printf("凶手是%c\n", killer);

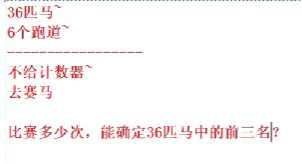
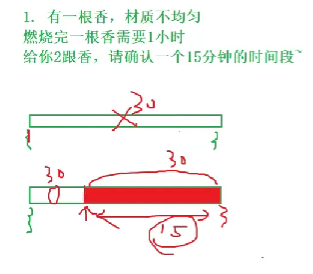
}

}

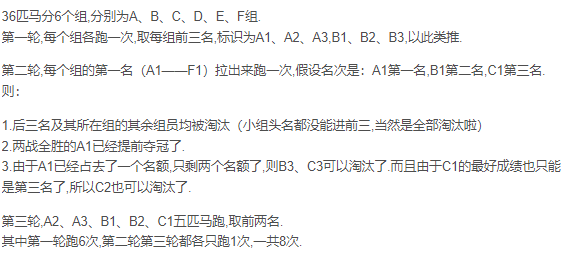
return 0;

}

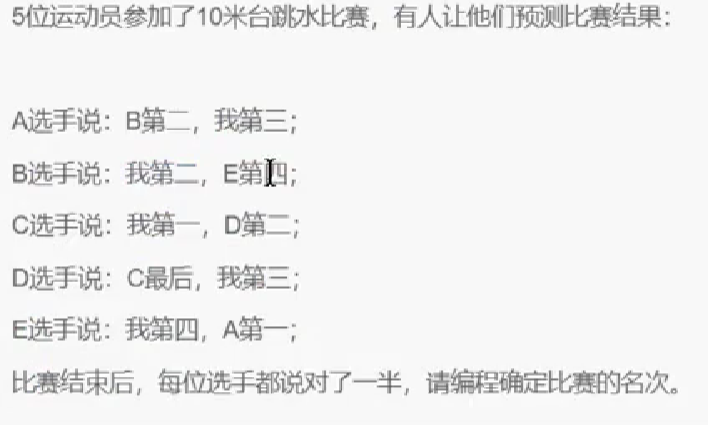
例：



解：



例：



解：#include<stdio.h>

int main()

{

int a = 0;

int b = 0;

int c = 0;

int d = 0;

int e = 0;

for (a = 1; a <= 5; a++)//每个人的名次都在1-5之间可能

{//穷举所有可能性，然后判断他们说的话的真假

for (b = 1; b <= 5; b++)

{

for (c = 1; c <= 5; c++)

{

for (d = 1; d <= 5; d++)

{

for (e = 1; e <= 5; e++)

{//每个人说对一半

if (((b == 2) + (a == 3) == 1)

&& ((b == 2) + (e == 4) == 1)

&& ((c == 1) + (d == 2) == 1)

&& ((c == 5) + (d == 3) == 1)

&& ((e == 4) + (a == 1) == 1))

{//这时的名次即为正确的名次

if (a \* b \* c \* d \* e == 120)//避免存在重复名次

{

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

}

}

}

}

}

}

}

return 0;

}

**二级指针传参：**

#include<stdio.h>

void test(int\*\* ppa)//二级指针接收

{

}

//二级指针传参

int main()

{

int arr[] = { 1,2,3,4,5 };

int\* pa = arr;

int\*\* ppa = &pa;

test(ppa);//或者一级指针的地址传参

int\* brr[10] = { 0 };

test(brr);//传指针数组的数组名

return 0;

}

5 函数指针

指向函数的指针，存放的是函数地址的指针，类比与前面的定义。

**注：数组名 ！= &数组名；而函数名 == &函数名。**

#include<stdio.h>

void test(int\* p)

{

}

int main()

{

int a = 0;

int\* pa = &a;//整型指针，指向整型的指针，存放整型变量的地址

char ch = 'w';

char\* pc = &ch;//字符指针，指向字符的指针，存放字符变量的地址

int arr[10] = { 0 };

int(\*parr)[10] = &arr;//数组指针，指向数组的指针，存放数组的地址

void (\*pf)(int\* ) = &test;//函数指针，指向函数的指针，存放函数的地址

return 0;

}

函数指针的解引用，作为函数进行调用，函数调用的三种方式。

#include<stdio.h>

int add(int x, int y)

{

return x + y;

}

int main()

{

//int (\*padd)(int, int) = &add;//padd是一个函数指针

int (\*padd)(int, int) = add;//add == &add == pf

//int ret = (\*padd)(3, 5);//1函数指针解引用得到函数，然后使用函数

//int ret = add(3, 5);//2

int ret = padd(3, 5);//3

printf("%d\n", ret);

return 0;

}

例1：

#include<stdio.h>

int main()

{

(\*(void (\*)())0)();//0是一个数字，或者编号，编号我们可以作为地址，从而和指针联系起来

//强制类型转换,void (\*)();//函数指针类型 前面\*号解引用，后面括号是函数调用

//调用0地址处的函数，将0当作函数的地址。

return 0;

}

例2：

#include<stdio.h>

#include<signal.h>

int main()

{

void(\*signal(int, void(\*)(int)))(int);//signal(int, void(\*)(int))函数名，函数参数，还缺少函数返回类型

//而去掉刚刚的部分还剩下void(\*)(int)，说明signal函数的返回类型为void(\*)(int)类型。

//这里只是函数signal的声明。

//void(\*)(int) signal(int, void(\*)(int));//本来函数声明应该写成这样，但这里语法不支持

//typedef void(\*)(int) pfun\_t;//函数指针类型重定义这样写也不行。

typedef void(\* pfun\_t)(int) ;//这样写才行

//那么上面的函数声明可以简化为

pfun\_t signal(int, pfun\_t);

return 0;

}

6 函数指针数组

函数指针数组，本质上是数组，数组元素为函数指针。

#include<stdio.h>

int add(int x, int y)

{

return x + y;

}

int sub(int x, int y)

{

return x - y;

}

int main()

{

int (\*pf1)(int, int) = add;//pf1,pf2是函数指针

int (\*pf2)(int, int) = sub;

int (\*pfarr[10])(int, int) = { pf1, pf2 };//pfarr[10]函数指针数组，注意固定搭配，去掉之后即为数据类型

return 0;

}

**例：利用函数指针写一个数的四则运算计算器。**

#include<stdio.h>

void menu()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\* 1.Add 2.Sub \*\*\*\*\*\*\n");

printf("\*\*\*\*\*\* 3.Mul 4.Div \*\*\*\*\*\*\n");

printf("\*\*\*\*\*\* 0.Exit \*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

float Add(float x, float y)

{

return x + y;

}

float Sub(float x, float y)

{

return x - y;

}

float Mul(float x, float y)

{

return x \* y;

}

float Div(float x, float y)

{

return x / y;

}

int main()

{

int input = 0;

float x = 0;

float y = 0;

float (\*pfarr[5])(float, float) = { NULL, Add, Sub, Mul, Div };//函数指针数组初始化 转移表

while (1)

{

menu();

printf("请选择：");

scanf("%d", &input);

if (input == 0)

{

printf("退出程序\n");

break;

}

else if (input >= 5 || input < 0)

{

printf("输入错误，请重新输入\n");

continue;

}

else

{

printf("请输入两个数：");

scanf("%f %f", &x, &y);

printf("%f\n", pfarr[input](x, y));//函数指针数组调用函数

}

}

return 0;

}

7 指向函数指针数组的指针

指向函数指针数组的指针，则表示指针，而是指向函数指针数组的指针，或者叫函数指针数组指针。

#include<stdio.h>

int Add(int x, int y)

{

return x + y;

}

int Sub(int x, int y)

{

return x - y;

}

int main()

{

int arr[10] = { 0 };

int(\*parr)[10] = &arr;//parr是数组指针

int (\*pfarr[5])(int, int) = { Add, Sub };//&pfarr表示函数指针数组的地址

int (\*(\*pfarrp)[5])(int, int) = &pfarr;//猜测函数指针数组指针的初始化和定义

return 0;

}

8 回调函数

比如将A函数的地址作为参数传递给B函数，B函数需要用一个函数指针来接收，当我们在使用B函数时，通过函数指针调用A函数的过程叫回调函数。

#include<stdio.h>

//回调函数实现计算器，将函数的地址作为参数传递到另一个函数的函数指针去调用的过程

void menu()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\*\*\*\*\*\* 1.Add 2.Sub \*\*\*\*\*\*\n");

printf("\*\*\*\*\*\* 3.Mul 4.Div \*\*\*\*\*\*\n");

printf("\*\*\*\*\*\* 0.Exit \*\*\*\*\*\*\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

float Add(float x, float y)

{

return x + y;

}

float Sub(float x, float y)

{

return x - y;

}

float Mul(float x, float y)

{

return x \* y;

}

float Div(float x, float y)

{

return x / y;

}

float calc(float(\*p)(float, float))

{

float x = 0;

float y = 0;

printf("请输入两个数：");

scanf("%f %f", &x, &y);

return p(x, y);

}

int main()

{

int input = 0;

float ret = 0;

do

{

menu();

printf("请选择：");

scanf("%d", &input);

switch (input)

{

case 0:

printf("退出程序\n");

break;

case 1:

ret = calc(Add);//将函数的地址作为参数调用函数，回调函数

printf("%f\n", ret);

break;

case 2:

ret = calc(Sub);

printf("%f\n", ret);

break;

case 3:

ret = calc(Mul);

printf("%f\n", ret);

break;

case 4:

ret = calc(Div);

printf("%f\n", ret);

break;

default:

printf("输入错误，请重新输入\n");

break;

}

} while (input);

return 0;

}

**例：qsort()函数的功能，快速排序函数，所有类型的参数都可以排序，与冒泡排序单一功能不同。**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//库函数-快速排序函数qsort()读参考文档，可以排序各种类型数组元素

//void qsort(void\* base, //base数组元素首地址，void\*表示无类型指针，方便传输各种类型数组

// size\_t num, //num数组元素个数

// size\_t width, //width元素所占字节数，由于指针是无类型的，所以必须说明元素内存大小

// int(\_\_cdecl\* compare)(const void\* elem1, const void\* elem2)

//);//compare函数指针，比较函数，参数是无类型指针，比较两个元素的大小，返回值>0、 =0、 <0

//1 整型数组排序

//数组打印函数

//void print\_arr(int arr[], int sz)

//{

// int i = 0;

// for (i = 0; i < sz; i++)

// {

// printf("%d ", arr[i]);

// }

// printf("\n");

//}

//

////整型元素比较函数

//int int\_cmp(const void\* e1, const void\* e2)

//{

// return \*((int\*)e1) - \*((int\*)e2);//原本e1和e2是无类型指针，则需要强制类型转换再解引用比较大小

//}//若需要降序，则将e1和e2反着写即可

//

////利用库函数qsort()实现整型数组排序

//void int\_arr\_sort(int arr[], int sz)

//{

// print\_arr(arr, sz);//排序之前的数组

// qsort(arr, sz, sizeof(arr[0]), int\_cmp);//对数组进行排序，函数名传递给函数指针，升序。

// print\_arr(arr, sz);//排序过后的数组打印

//}

//2 结构体数组按年龄排序

struct stu

{

char name[10];

int age;

};

//结构体数组元素比较函数-按年龄比较

//int stru\_cmp(const void\* e1, const void\* e2)

//{

// return ((struct stu\*)e1)->age - ((struct stu\*)e2)->age;//强制类型转换

//}

//3 结构体数组按姓名排序

int stru\_cmp(const void\* e1, const void\* e2)

{

return strcmp(((struct stu\*)e1)->name, ((struct stu\*)e2)->name);//强制类型转换

}//字符串比较函数是将字符串的每一位的ASCII码比较的，前位越大的则字符串越大

//结构体数组排序函数

void stru\_arr\_sort(struct stu sarr[], int sz)

{

qsort(sarr, sz, sizeof(sarr[0]), stru\_cmp);

}

//主函数

int main()

{

//1 整型数组排序

/\*int arr[10] = { 0,9,6,4,5,7,8,3,2,1 };

int sz = sizeof(arr) / sizeof(arr[0]);

int\_arr\_sort(arr, sz);\*/

//2 结构体数组排序

struct stu sarr[] = { {"zhangsan", 30},{"lisi", 34},{"wangwu", 20} };

int sz = sizeof(sarr) / sizeof(sarr[0]);

stru\_arr\_sort(sarr,sz);

return 0;

}

**例：以冒泡排序思想实现qsort()函数的功能，实现各种数组的排序。**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//库函数-快速排序函数qsort()读参考文档，可以排序各种类型数组元素

//void qsort(void\* base, //base数组元素首地址，void\*表示无类型指针，方便传输各种类型数组

// size\_t num, //num数组元素个数

// size\_t width, //width元素所占字节数，由于指针是无类型的，所以必须说明元素内存大小

// int(\_\_cdecl\* compare)(const void\* elem1, const void\* elem2)

//);//compare函数指针，比较函数，参数是无类型指针，比较两个元素的大小，返回值>0、 =0、 <0

//1 整型数组排序

//数组打印函数

//void print\_arr(int arr[], int sz)

//{

// int i = 0;

// for (i = 0; i < sz; i++)

// {

// printf("%d ", arr[i]);

// }

// printf("\n");

//}

//

////整型元素比较函数

//int int\_cmp(const void\* e1, const void\* e2)

//{

// return \*((int\*)e2) - \*((int\*)e1);//原本e1和e2是无类型指针，则需要强制类型转换再解引用比较大小

//}//若需要降序，则将e1和e2反着写即可

//

//交换函数Swap()

void Swap(char\* e1, char\*e2, int width)

{

int i = 0;

for (i = 0; i < width; i++)//每一个字节进行交换

{

char tmp = \*e1;

\*e1 = \*e2;

\*e2 = tmp;

e1++;

e2++;

}

}

//

//以冒泡排序的思想实现qsort()函数的功能

void bubble\_sort(void\* base, int sz, int width, int (\*cmp)(const void\* e1, const void\* e2))

{

int i = 0;

int j = 0;

//冒泡排序

for (i = 0; i < sz - 1; i++)//冒泡排序趟数

{

for (j = 0; j < sz - 1 - i; j++)//每趟比较的次数

{

//什么条件下交换

if (cmp((char\*)base + j \* width, (char\*)base + (j + 1) \* width) > 0)

//传元素的地址，强制类型转换为char\*

//当比较函数返回值大于零时交换，比较函数的参数传递

{

//交换函数

Swap((char\*)base + j \* width, (char\*)base + (j + 1) \* width, width);

//由于不知道元素大小，所以交换时以每个字节进行交换，需要传递元素大小

}

}

}

}

//

////利用自定义函数bubble\_sort()实现整型数组排序

//void int\_arr\_sort(int arr[], int sz)

//{

// print\_arr(arr, sz);//排序之前的数组

// bubble\_sort(arr, sz, sizeof(arr[0]), int\_cmp);//对数组进行排序，函数名传递给函数指针，升序。

// print\_arr(arr, sz);//排序过后的数组打印

//}

//

//

////主函数

//int main()

//{

// //1 整型数组排序

// int arr[10] = { 0,9,6,4,5,7,8,3,2,1 };

// int sz = sizeof(arr) / sizeof(arr[0]);

// int\_arr\_sort(arr, sz);

//

// return 0;

//}

//2 结构体数组按年龄排序

struct stu

{

char name[10];

int age;

};

//结构体数组元素比较函数-按年龄比较

//int stru\_cmp(const void\* e1, const void\* e2)

//{

// return ((struct stu\*)e1)->age - ((struct stu\*)e2)->age;//强制类型转换

//}

//3 结构体数组按姓名排序

int stru\_cmp(const void\* e1, const void\* e2)

{

return strcmp(((struct stu\*)e1)->name, ((struct stu\*)e2)->name);//强制类型转换

}//字符串比较函数是将字符串的每一位的ASCII码比较的，前位越大的则字符串越大

//结构体数组排序函数

void stru\_arr\_sort(struct stu sarr[], int sz)

{

bubble\_sort(sarr, sz, sizeof(sarr[0]), stru\_cmp);

}

int main()

{

//2 结构体数组排序

struct stu sarr[] = { {"zhangsan", 30},{"lisi", 34},{"wangwu", 20} };

int sz = sizeof(sarr) / sizeof(sarr[0]);

stru\_arr\_sort(sarr, sz);

return 0;

}

注：只有sizeof(数组名)和&数组名指的是整个数组，其余的数组名均代表首元素地址，二维指针数组名代表二维数组首元素地址，即第一行元素地址。Strlen()函数是以地址传参的，且会将地址强制类型转换为char\*，然后通过起始地址向后访问，当遇到0或’\0’停止计算，且’\0’不计入长度。

例：

#include<stdio.h>

int main()

{//分析以下代码的输出

int a[3][4] = { 0 };

printf("%d\n", sizeof(a));//48 a为二维数组数组名，且与sizeof单独放，则此时a代表的是整个数组，计算的是整个数组的大小.

printf("%d\n", sizeof(a[0][0]));//4 a[0][0]表示第一行第一列元素.

printf("%d\n", sizeof(a[0]));//16 a[0][i]表示的第一行元素,a[0]表示的是第一行一维数组名，但与sizeof单独放，则表示整个第一行一维数组，计算的是第一行的大小.

printf("%d\n", sizeof(a[0] + 1));//4/8 由上，a[0]表示第一行一维数组名，但并没有单独放在sizeof内，则只表示第一行数组首地址，a[0]+1代表第一行第二个元素的地址.

printf("%d\n", sizeof(\*(a[0] + 1)));//4 由上，a[0] + 1表示第一行第二个元素的地址，则\*(a[0] + 1)代表该元素.

printf("%d\n", sizeof(a + 1));//4/8 a为二维数组名，代表的是二维数组首地址,即第一行元素地址，a+1表示第二行元素地址.

printf("%d\n", sizeof(\*(a + 1)));//16 由上，a+1代表第二行元素地址则\*(a+1)代表第二行元素.

printf("%d\n", sizeof(&a[0] + 1));//4/8 由上，a[0]表示第一行数组名，与&一起代表第一行元素的地址，&a[0] + 1代表第二行元素地址.

printf("%d\n", sizeof(\*(&a[0] + 1)));//16 由上，&a[0] + 1代表第二行元素地址,\*(&a[0] + 1))代表第二行元素.

printf("%d\n", sizeof(\*a));//16 由上，a表示第一行元素地址，\*a代表第一行元素.

printf("%d\n", sizeof(a[3]));//16 a[3]本来是越界访问了，但sizeof只关心参数的类型属性，内部表达式不进行计算，如果进行计算，a[3]的属性是第四行数组名，与sizeof单独放，计算的是第四行的大小.

return 0;

}

习题1：

#include<stdio.h>

struct stu

{

char name[5];

int age;

int score[4];

}\*p;//假设整个结构体大小为n个字节空间，且p的值为0x...

int main()

{

//读程序

printf("%p\n", p + 0x1);//0x1其实就是1，p为结构体指针，指针类型

//决定了指针访问的内存大小，p+1跳过一个结构体大小0x...+n字节

printf("%p\n", (unsigned long)p + 0x1);//将p转换为long型，则加一

printf("%p\n", (unsigned int\*)p + 0x1);//将p转换为int\*型，则加一跳过四个字节

return 0;

}

习题2：

#include<stdio.h>

int main()

{

int a[4] = { 1,2,3,4 };

int\* ptr1 = (int\*)(&a + 1);//0x00000004

int\* ptr2 = (int\*)((int)a + 1);//0x02000000 (int)地址+1表示

//地址加一个字节，小端存储，内存仔细分析。

printf("%x, %x\n", ptr1[-1], \*ptr2);//%x十六进制打印有效位4，2000000

return 0;

}

习题3：

#include<stdio.h>

int main()

{

int a[3][2] = { (0,1),(2,3),(4,5) };//逗号表达式，内容1，3，5

int\* p = a[0];//a[0]表示第一行数组名即第一行元素首地址

printf("%d\n", p[0]);//p[i] == \*(p + i)

return 0;

}

习题4：

#include<stdio.h>

int main()

{

int a[5][5];//5行5列的二维数组

int(\*p)[4];//p是一个指向含有四个整型的数组指针

p = a;//a为二维数组第一行数组地址五个元素，但p只指向四个元素的数组

printf("%p, %d\n", &p[4][2] - &a[4][2], &p[4][2] - &a[4][2]);

//&p[4][2]=&(\*(\*(p+4)+2)),p为数组指针加1跳过一个4元素的数组

//通过内存分析，可知&p[4][2] - &a[4][2]=-4，以%d打印为-4，

//如果以%p打印，则认为内存中的补码为地址打印出来FFFFFC

return 0;

}

习题5：自己分析，注意**加1和减1操作会改变变量本身**。

#include<stdio.h>

int main()

{

char\* c[] = { "ENTER","NEW","POINT","FIRST" };

char\*\* cp[] = { c + 3,c + 2,c + 1,c };

char\*\*\* cpp = cp;

printf("%s\n", \*\*++cpp);//POINT

printf("%s\n", \*-- \* ++cpp + 3);//ER

printf("%s\n", \*cpp[-2] + 3);//ST

printf("%s\n", cpp[-1][-1] + 1);//EW

return 0;

}

练习题：

（1）当我们用free释放指针时，指针的内容置为NULL（并非自动置位）。

解：动态内存开辟，先申请内存malloc，int\* p = malloc(40)，然后使用内存，最后要进行内存释放，free(p)//释放给操作系统，然后手动置位p = NULL.，而非自动置位。

（2）杨氏矩阵：有一个数字矩阵，矩阵的每行从左到右是依次递增的，矩阵从上到下是递增的，请编写程序查找某个数字在矩阵中是否存在。要求：时间复杂度小于O(N).

自己写的：

#include<stdio.h>

//杨氏矩阵：有一个数字矩阵，矩阵的每行从左到右是依次递增的，

//矩阵从上到下是递增的，请编写程序查找某个数字在矩阵中是否存在。

//要求：时间复杂度小于O(N).

int int\_search(int arr[10][10], int search)

{

int i = 0;

int j = 0;

int left = 0;//左下标

int right = 9;//右下标

int mid = 0;

mid = (left + right) / 2;//中心位置

//时间复杂度小于N，则由于每行递增，可以采取每一行进行二分查找。

//时间复杂度nlog2n < n\*n = N

for (i = 0; i < 10; i++)

{

while (left <= right)//每一行

{

if (search == arr[i][mid])//二分查找

{

printf("行 = %d, 列 = %d\n", i, mid);

return 1;//找到了返回

}

else if (search < arr[i][mid])

{

right = mid - 1;

}

else

{

left = mid + 1;

}

mid = (left + right) / 2;//改变中心位置

}

right = 9;//每行结束，重新设置下标

left = 0;

mid = (left + right) / 2;

}

return 0;//没找到返回

}

int main()

{

int arr[10][10] = { 0 };

int i = 0;

int j = 0;

int count = 0;

int search = 0;

for (i = 0; i < 10; i++)

{

for (j = 0; j < 10; j++)

{

arr[i][j] = ++count;

printf("%d ", arr[i][j]);

}

printf("\n");

}

scanf("%d", &search);

if (int\_search(arr, search) == 1)

{

printf("找到了\n");

}

else

{

printf("没找到\n");

}

return 0;

}

**鹏哥分析：**每一行的右上角元素与查找元素进行比较，如果查找元素大于右上角元素，则说明不在此行，跳过这一行；如果查找元素小于右上角元素，则跳过该列；循环查找，直到找到或者不满足循环条件没找到。

#include<stdio.h>

int int\_search(int arr[10][10], int search, int\* px, int\* py)

{

int x = 0;

int y = 9;

while (x < \*px && y >= 0)

{//查找元素与左上角元素比较

if (search > arr[x][y])

{

x++;

}

else if (search < arr[x][y])

{

y--;

}

else

{

\*px = x;

\*py = y;

return 1;

}

}

return 0;

}

int main()

{

int arr[10][10] = { 0 };

int i = 0;

int j = 0;

int x = 10;//传址调用，使得坐标被返回

int y = 10;

int count = 0;

int search = 0;

for (i = 0; i < 10; i++)

{

for (j = 0; j < 10; j++)

{

arr[i][j] = ++count;

printf("%d ", arr[i][j]);

}

printf("\n");

}//创建数组

scanf("%d", &search);

if (int\_search(arr, search, &x, &y) == 1)

{

printf("找到了\n");

printf("坐标为：%d %d\n", x, y);

}

else

{

printf("没找到\n");

}

return 0;

}

（3）左旋字符串，如ABCDEF，左旋两个变成CDEFAB。

#include<stdio.h>

#include<string.h>

//左旋字符串，如ABCDEF，左旋两个变成CDEFAB。

//左边的字母换到右边，其余朝左边移位

void string\_left\_rotate(char\* str, int n)

{

//左旋多少个字符就进行多少次换位移位

int i = 0;

int j = 0;

int len = strlen(str);

char tmp = 0;

for (i = 0; i < n; i++)

{

//1.首字母放到缓存

tmp = \*str;

//2.后面的字母依次往前一一个内存

for (j = 0; j < len - 1; j++)

{

str[j] = str[j + 1];

}

//3.首字母放到最后

str[len - 1] = tmp;

}

}

int main()

{

char str[] = "ABCDEFG";

int n = 0;

scanf("%d", &n);

string\_left\_rotate(str, n);

printf("%s\n", str);

return 0;

}

（4）写一个函数，判断一个字符串是否为另一个字符串旋转之后的字符串，包括左旋或右旋，也包括多次旋转。其实左旋与右旋是一组类似逆运算的，如果某一字符串左旋len次仍与另一字符串不同，则说明两者不可以通过旋转的到。

#include<stdio.h>

#include<string.h>

//左旋字符串，如ABCDEF，左旋两个变成CDEFAB。

//左边的字母换到右边，其余朝左边移位

//左旋函数

void string\_left\_rotate(char\* str, int n)

{

//左旋多少个字符就进行多少次换位移位

int i = 0;

int j = 0;

int len = strlen(str);

char tmp = 0;

for (i = 0; i < n; i++)

{

//1.首字母放到缓存

tmp = \*str;

//2.后面的字母依次往前一一个内存

for (j = 0; j < len - 1; j++)

{

str[j] = str[j + 1];

}

//3.首字母放到最后

str[len - 1] = tmp;

}

}

int Is\_string\_rotate\_same(char str1[], char str2[])

{

int len = strlen(str1);

int i = 0;

char str\_tmp[10];//字符串缓存区

strcpy(str\_tmp, str1);

for (i = 0; i < len; i++)

{

string\_left\_rotate(str\_tmp, i);//循环左旋，看是否可以通过旋转而来

if (strcmp(str\_tmp, str2) == 0)

{

return 1;

}

strcpy(str\_tmp, str1);//重新复位

}

return 0;

}

int main()

{

char str1[] = "ABCDEFG";

char str2[] = "DEFGABC";

if (Is\_string\_rotate\_same(str1, str2) == 1)

{

printf("str1可以由str2旋转而来\n");

}

else

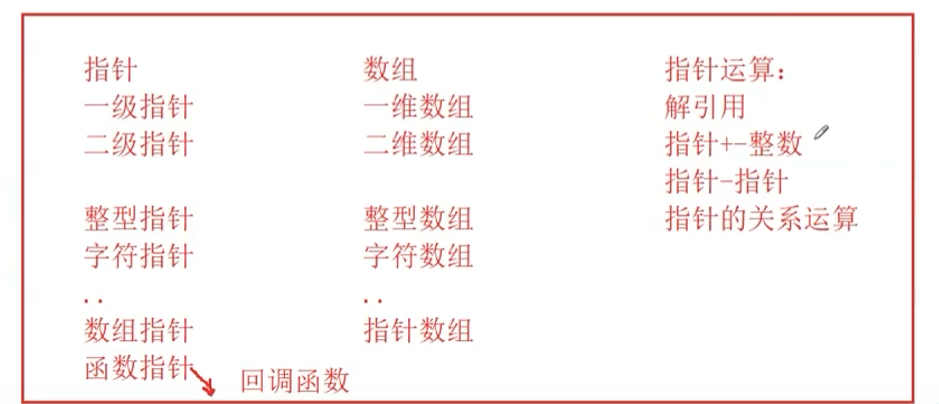
{

printf("str1不可以由str2旋转而来\n");

}

return 0;

}



9 指针和数组面试题的解析