

**课程 实 验 报 告**

**课程名称： C语言程序设计课程实验**

**专业班级： 自实1901**

**学 号： U201916457**

**姓 名： 张皓然**

**指导教师：**

**报告日期： 2020年4月13日**

**人工智能与自动化学院**

# 实验1 1、 用实例说明一维数组、二维数组、数组指针，在函数间的传递方法，用一个表总结实参、形参的对应关系；用表格总结一级指针、数组指针和二级指针访问目标数据的各种形式。

## 实验目的

（1）掌握一维数组，二维数组，数组指针在函数间传递的方法。

（2）归纳总结他们实参形参的对应关系。

（3）归纳总结一级指针，数组指针，二级指针访问目标数据的各种形式。

## 设计思路与代码实现

### 设计思路

先分别定义初始化一维数组，二维数组，数组指针。然后将他们传递到一个函数中输出。

### 代码实现

#include<stdio.h>

void print(int a1[], int a2[][3], int\* pa[], int n1, int n2);

int main()

{

int a1[4] = { 1,2,3,4 };

int a2[2][3] = { {1,2,3},{4,5,6} };

int\* pa[2];

pa[0] = a2[0];

pa[1] = a2[1];

print(a1, a2, pa, 4, 2);

return 0;

}

void print(int\* a1, int(\*a2)[3], int\* pa[], int n1, int n2)

{

int i, j;

for (i = 0; i < n1; i++)

{

printf("%d\t", a1[i]);

}

printf("\n");

for (i = 0; i < n2; i++)

{

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

{

printf("%d\t", a2[i][j]);

}

printf("\n");

}

for (i = 0; i < n2; i++)

{

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

{

printf("%d\t", pa[i][j]);

}

printf("\n");

}

}

## 实验过程

（1）编写程序，编译运行（CTRL\_F9）

（2）在用户界面查看运算结果（ALT\_F5）

（3）分析实验结果，总结归纳

## 实验结果

1 2 3 4

1 2 3

4 5 6

1 2 3

4 5 6

实验结果符合预期，从函数声明与函数定义处的写法可以看出，一维数组，二维数组，指针数组在函数间的传递依旧遵循指针与数组的互换性。

## 总结表格

一维数组，二维数组，指针数组在函数间传递的形参与实参对应表

|  |  |  |
| --- | --- | --- |
|  | 实参 | 形参 |
| 一维数组int a[10] | a, &a[0] | int a[], int\* a |
| 二维数组int a[2][3] | a, &a[0] | int a[][3], int (\*a)[3] |
| 指针数组int\* a[2] | a | int\* a[] |

一级指针，数组指针，二级指针访问目标数据的各种形式(默认指向对应数组)

|  |  |
| --- | --- |
| 一级指针int\* a | \*(a+i), a[i] |
| 数组指针int (\*a)[3] | a[i][j], \*(a[i]+j), \*(\*(a+i)+j) |
| 二级指针int\*\* a(指向指针数组) | a[i][j], \*(a[i]+j), \*(\*(a+i)+j) |

# 实验2 总结指针使用过程中常见的错误，用实例进行说明。

## 实验目的

总结指针使用过程中的常见错误

## 设计思路与代码实现

### 设计思路

在代码中融入指针使用过程中的常见错误

### 代码实现

#include<stdio.h>

int main()

{

register int a = 1;

register int\* pa = NULL;

int b[3] = { 1,2,3 };

int\* pb;

pb = b - 1;

pb++;

b++;

pa = &a;

printf("%d %d %d %d", \*pa, \*pb, \*b, \*pb++);

return 0;

}

## 实验过程

（1）编写代码，编译运行程序

（2）运行失败，利用单步调试的技巧分析代码的问题（F4，ALT\_F4，F7）

（3）持续调试，直至程序运行成功

## 实验结果

（1）初始时运行失败

（2）在调试后将程序改为，

#include<stdio.h>

int main()

{

int a = 1;

int\* pa = NULL;

int b[3] = { 1,2,3 };

int\* pb;

pb = b - 1;

pb++;

pa = &a;

printf("%d %d %d %d", \*pa, \*pb, \*b, \*pb++);

}

（3）更改后再次运行，结果为，

1 2 1 1

与实验预期相符。

## 实验分析

在指针初始化时，不能将指针指向register型变量。这是因为指针只能指向内存中的地址，而register型变量不存储在内存中。有趣的是，我们可以定义一个register型的指针。

数组虽是指针，但他们是指针常量。如指针pa可以进行自增操作，操作后pa即指向地址中下一位的元素。然而对于数组a[10]，a是地址常量，改变自身值的事情，他无法实现。

取地址符和自增运算符是同级运算符，执行时遵循从右到左的顺序。即\*pa++，先执行pa++,再执行\*pa。不过由于pa++是运算后再执行，所以这条语句的实际效果仍旧是先对pa取内容，然后pa自增。而\*++pa，按照从右往左的顺序，依旧是先执行自增运算，此时++pa会先执行。所以这条语句的实际效果是，先对pa自增，然后再执行\*pa。但此举会改变pa的指向，不利于后续对pa的操作。

# 实验3 指针型函数，返回的地址有什么限制？用实例说明注意的事项。

## 实验目的

掌握指针型函数返回的地址的要求

## 设计思路与代码实现

### 设计思路

参考书上关于计算平方的例子

### 代码实现

#include<stdio.h>

double\* square(double);

int main()

{

double num = 5.0;

double\* ptr = 0;

ptr = square(num);

printf("num's square = %f\n", num \* num);

printf("result = %f\n", \*ptr);

return 0;

}

double\* square(double data)

{

double result = 0.0;

result = data \* data;

return &result;

}

## 实验过程

（1）编写代码，编译运行程序

（2）运行失败，结果与预期不符，利用单步调试的技巧分析代码的问题（F4，ALT\_F4，F7）

（3）持续调试，直至程序运行成功

## 实验结果

（1）

num's square = 25.000000

result = -92559631349317830736831783200707727132248687965119994463780864.000000

与预期不符，进行调试。

（2）将函数中result的定义改为static double result = 0.0;

（3）更改后的结果为

num's square = 25.000000

result = 25.000000

## 实验分析

指针型函数的返回值是一个变量的地址。那就要保证，当函数将这个地址返回给主函数后，在主函数中调用这个地址的目标数据时，目标数据依旧存在。即函数运行结束后，地址中的数据不会从内存中消失。这就要求返回的指针所指向的数据，不是自动型变量和寄存器变量，可以是静态变量或是全局变量。

# 实验4 求两个整数之间的所有素数，要求函数返回所有的素数，主函数调用并显示所有素数

## 实验目的

求两个整数间的所有素数，函数返回所有素数，并在主函数中输出

## 设计思路与代码实现

### 设计思路

（1）设计函数原型：因为需要向主函数中传递不止一个数据，所以考虑利用数组。函数原型可以设计为int sushu(int\*,int,int,int) int\*传递一个数组进入函数，一个int用于确定数组的元素个数，剩余两个int是输入的两个整数。

（2）设计函数：流程图大致如下

判断a,b的大小

max=a,min=b

max=b,min=a

a大

b大

从min直至max

循环

min<t<max

用遍历的方法判断t

是否是素数

将t装入数组对应位置

无操作

（3）设计main函数用于调用测试。

### 代码实现

#include<stdio.h>

int sushu(int\*, int, int, int);

int main()

{

int a[100];

int x, y;

int num, i;

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

num = sushu(a, x - y, x, y);

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

{

if (i % 5 == 0)

{

printf("\n");

}

if (a[i] == 1 || a[i] == 0)

{

continue;

}

printf("%d\t", a[i]);

}

return 0;

}

int sushu(int\* pa, int n, int x, int y)

{

int t;

int i = 0, j;

int sign = 0;

if (n < 0)

{

n = -n;

t = x;

x = y;

y = t;

}

y++;

while(y < x)

{

for (j = 2; j < y; j++)

{

if (y % j == 0)

{

sign = 1;

break;

}

}

if (sign == 0)

{

pa[i] = y;

i++;

}

y++;

sign = 0;

}

return i;

}

## 测试集

|  |  |
| --- | --- |
| 第一组 | 1 2 |
| 第二组 | 3 40 |
| 第三组 | 5 10 |
| 第四组 | 0 20 |

## 实验过程

（1）编写程序，编译运行（CTRL\_F9）

（2）在用户界面查看运算结果（ALT\_F5）

（3）分析实验结果，总结归纳

## 实验结果

第一组：

（无输出）

第二组：

5 7 11 13 17

19 23 29 31 37

第三组：

7

第四组：

2 3 5 7

11 13 17 19

## 实验分析

对于一个数是否是素数的判断，遍历判断是最常用的一种方法。除此以外，我们也可以将奇数偶数先分类，偶数一定不是素数，用遍历法判断（奇数/2）是不是素数。

# 实验5 编写程序，对整型数组进行升序排序，要求排序函数具有根据所带参数，决定是采用选择法还是冒泡法。

## 实验目的

编写程序，对整型数组内的数据排序。额外用一个参数判断是选择排序法还是冒泡排序法。

## 设计思路与代码实现

### 设计思路

设计两个函数，一个函数是选择排序法 ，一个函数是冒泡排序法。在主函数中进行一次判断，然后分类进行选择排序或是冒泡排序法。

sign是否为0

选择排序法

冒泡排序法

是

否

### 代码实现（以a[5]为例）

#include<stdio.h>

void xuanze(int\*, int);

void maopao(int\*, int);

int main()

{

int a[5];

int i;

int sign;

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

{

scanf\_s("%d", &a[i]);

}

scanf\_s("%d", &sign);

if (sign == 0)

{

xuanze(a, 5);

}

else {

maopao(a, 5);

}

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

{

printf("%d\t", a[i]);

}

return 0;

}

void xuanze(int\* pa, int n)

{

int i, j;

int t;

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

{

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

{

if (pa[i] > pa[j])

{

t = pa[i];

pa[i] = pa[j];

pa[j] = t;

}

}

}

}

void maopao(int\* pa, int n)

{

int i, j;

int t;

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

{

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

{

if (pa[j] > pa[j + 1])

{

t = pa[j];

pa[j] = pa[j + 1];

pa[j + 1] = t;

}

}

}

}

## 测试集

|  |  |  |
| --- | --- | --- |
| 组别 | sign | 数组 |
| 第一组 | 1 | 1 5 6 2 1 |
| 第二组 | 2 | 0 4 10 5 2 |
| 第三组 | 3 | 6 9 -1 5 20 |
| 第四组 | 0 | 9 8 76 5 4 |

## 实验过程

（1）编写程序，编译运行（CTRL\_F9）

（2）在用户界面查看运算结果（ALT\_F5）

（3）分析实验结果，总结归纳

## 实验结果

第一组

1 1 2 5 6

第二组

0 2 4 5 10

第三组

-1 5 6 9 20

第四组

4 5 8 9 76

## 实验分析

选择排序法i从0到n-1（最后一位不必判断），j从i到n（每轮确定上一轮i的位置）。冒泡排序法i从0到n-1（首位不必判断），j从0到n-1-i（每轮确定倒数第i位）

# 实验6上机验证6.17，6.24，6.25，6.26

## 实验目的

上机验证书上的四个例子，熟悉数组与指针的运用

## 例子6.17

### 代码

#include<stdio.h>

int main()

{

int num[3][4] = { {0,1,2,3},{1,2,3,4},{2,3,4,5} };

int\* p[3], \*\* pp;

int i, j;

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

{

p[i] = num[i];

}

pp = p;

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

{

for (j = 0; j < 4; j++)

{

printf("%-3d", \*(\*(pp + i) + j));

}

printf("\n");

}

return 0;

}

### 代码流程分析

定义二维数组

用指针数组，依次指向二维数组的每一行

二级指针指向指针数组的地址

利用二级指针

调用数组

### 实验过程

（1）输入程序，编译运行（CTRL\_F9）

（2）在用户界面查看运算结果（ALT\_F5）

（3）分析实验结果，进行逐步调试（F4，ALT\_F4，F7），了解程序运行过程。

### 实验结果

0 1 2 3

1 2 3 4

2 3 4 5

## 例6.24

### 代码

#include<stdio.h>

#define N1 3

#define N2 4

void convert(int(\*)[N2], int(\*)[N1], int, int);

int main()

{

int arr1[N1][N2] = { 1,2,3,4,5,6,7,8,9,10,11,12 };

int arr2[N2][N1];

int(\*p)[N2], (\*pt)[N1];

int i, j;

p = arr1;

pt = arr2;

convert(p, pt, N1, N2);

printf("the new array is:\n");

for (i = 0; i < N2; i++)

{

for (j = 0; j < N1; j++)

{

printf("%-4d", \*(\*(pt + i) + j));

}

printf("\n");

}

printf("the new array is:\n");

for (i = 0; i < N2; i++)

{

for (j = 0; j < N1; j++)

{

printf("%-4d", arr2[i][j]);

}

printf("\n");

}

return 0;

}

void convert(int(\*a)[N2], int(\*at)[N1], int row, int col)

{

int i, j;

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

{

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

{

\*(\*(at + j) + i) = \*(\*(a + i) + j);

}

}

}

### 代码流程分析

定义两个二维数组，对其中一个二维数组初始化

定义两个数组指针，分别指向刚刚定义的两个数组

将两个数组指针和N1，N2传入convert函数

把a的行列交换后转给at

以指针方式输出

以数组方式输出

### 实验过程

（1）编写程序，编译运行（CTRL\_F9）

（2）在用户界面查看运算结果（ALT\_F5）

（3）分析实验结果，总结归纳

### 实验结果

the new array is:

1 5 9

2 6 10

3 7 11

4 8 12

the new array is:

1 5 9

2 6 10

3 7 11

4 8 12

## 例6.25

### 代码

#include <stdio.h>

#define FALSE 0

#define TRUE !0

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int ascending(int, int);

int descending(int, int);

void exchange(int\*, int\*);

void sort(int[], int, int);

int main()

{

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

int flag1;

int i;

printf("please input num 0 or 1, 0=descending, 1=ascending:");

scanf\_s("%d", &flag1);

sort(num, 10, flag1);

printf("sorted array is:");

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

printf("%-4d", num[i]);

return 0;

}

void sort(int a[], int n, int flag)

{

int (\*ad)(int, int);

int t, c;

if (flag == TRUE)

ad = ascending;

else ad = descending;

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

for (c = 0; c < n - 1; c++)

if ((\*ad)(a[c], a[c + 1]))

exchange(&(a[c]), &(a[c + 1]));

}

int ascending(int a, int b) { return a > b; }

int descending(int a, int b) { return a < b; };

void exchange(int\* a, int\* b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

### 代码流程分析（sort函数）

定义函数指针

判断flag的值

函数指针指向升序函数

函数指针指向降序函数

非0

是0

冒泡法排序

### 实验过程

（1）输入程序，编译运行（CTRL\_F9）

（2）在用户界面查看运算结果（ALT\_F5）

（3）分析实验结果，总结归纳

### 实验结果

输入1：sorted array is:1 2 3 4 5 6 7 8 9 10

输入0：sorted array is:10 9 8 7 6 5 4 3 2 1

## 例6.26

### 代码

#include <stdio.h>

#include <stdlib.h>

#define N1 15

#define N2 5

void getrand(int\*, int);

void getsum(int\*, int\*, int);

void printarr(int\*, int);

int main()

{

int x[N1], w[N1 / N2] = { 0 };

getrand(x, N1);

printf("the number is:\n");

printarr(x, N1);

getsum(x, w, N1);

printf("the sum number is:\n");

printarr(w, N1 / N2);

return 0;

}

void getrand(int\* a, int n)

{

int i;

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

\*(a + i) = rand() % 100;

}

void printarr(int\* a, int n)

{

int i;

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

printf("%4d", \*(a + i));

printf("\n");

}

void getsum(int\* a, int\* b, int n)

{

int i, j = 0, sum = 0;

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

{

sum += \*(a + i);

if ((i + 1) % 5 == 0)

{

b[j] = sum; sum = 0; j++;

}

}

}

### 代码流程分析

该代码包含3个函数：随机赋值函数getrand，输出函数printarr，分块求和函数getsum。这三个函数都比较简单，只需要注意随机数的产生方法即可。

### 实验过程

（1）输入程序，编译运行（CTRL\_F9）

（2）在用户界面查看运算结果（ALT\_F5）

（3）分析实验结果，总结归纳

### 实验结果

the number is:

41 67 34 0 69 24 78 58 62 64 5 45 81 27 61

the sum number is:

211 286 219

附录

## 开发环境

本次实验中使用的环境配置如下：

（1）操作系统版本：Arch Linux x86\_64

（2）编译器及其版本：GCC version 8.1.1

（3）自动编译工具：CMake version 3.11.4

（4）编程环境：Borland C++ 3.1