C语言进阶学习

1. 分支与选择结构

1.1 前言

C语言是结构化的程序设计语言，包括顺序结构、选择结构、循环结构，与生活的规律相似。

1.2 分支语句

1.2.1 if语句

分号代表一句语句，所以结构的结束点要注意。if语句的结构：

int a = 6;

if (a == 6)

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

else

printf("出错啦\n");

C语言非零表示真，零表示假。

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<stdio.h>

int main()

{

int age = 0;

int i = 0;

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

{

printf("请输入年龄：\n");

scanf("%d", &age);

if (age < 18)

printf("未成年\n");

else if (age < 26)

printf("青年\n");

else if (age < 40)

printf("壮年\n");

else if (age < 60)

printf("中年\n");

else if (age < 100)

printf("老年\n");

else

printf("老不死\n");

}

return 0;

}

if else匹配问题，else与它最近的if匹配，代码的格式化。

1.2.2 switch语句

多分支语句，

switch(整型表达式)

{

case 整型常量表达式:

语句;

break;//多分支结构需要跳出顺序执行

}

int a = 0;

scanf("%d", &a);

switch (a)

{

case 0:

printf("出错了\n");

break;

case 1:

printf("星期一\n");

break;

case 2:

printf("星期二");

break;

default://默认情况，其余情况处理

printf("输入错误\n");

break;

}

case决定入口，break决定出口，default决定异常情况

1.3 循环结构

1.3.1 while语句（初始化条件、判断部分、条件改变部分三部分太开）

注意改变约束条件，break，跳出本层操作；continue跳出本次操作。

int ch = 0;

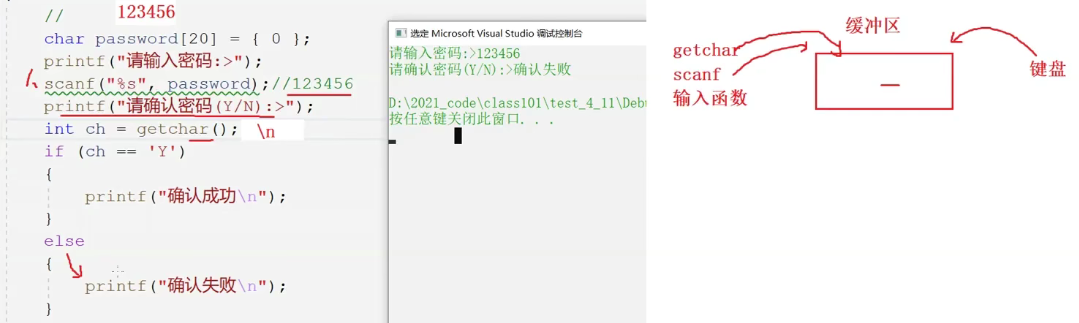
ch = getchar();//获取一个字符，返回它的ASCII码值整型

putchar(ch);

EOF文件结束标志，或者读取失败标志。//ctrl+z读取结束

while((ch=getchar())!=EOF)//成功获取

**注意：getchar()和scanf()输入函数输入字符串时**，（回车\n也属于字符数据scanf不获取（会存在缓冲区），而**getchar会获取**）获取数据是从缓冲区里面获取，当缓冲区里没有数据则等待键盘输入，若有，则直接获取，**尽量清空**。



#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<stdio.h>

int main()

{

int ch = 0;

char password[20] = { 0 };

printf("请输入密码：");

scanf("%s", password);

printf("请确认密码：（Y/N）");

//getchar();//清空缓存区的回车符

while (getchar() != '\n');//清空缓存区，不断获取等待。;相当于空语句。

ch = getchar();

if (ch == 'Y')

printf("确认成功\n");

else

printf("确认失败\n");

return 0;

}

1.3.2 for循环

for(表达式1；表达式2；表达式3)，初始化部分，判断部分，调整部分，循环体内改变循环变量会导致循环失去控制。判断条件采用左闭右开方式较好。最好不将表达式省略，可能会带来逻辑错误。break和continue。

1.3.3 do while循环

int main()

{

int i = 0;

do

{

printf("%d ", i);

i++;

} while (i <= 9);

return 0;

}

continue和break

**练习1：**n的阶乘和n的各项阶乘和。

//方法一

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<stdio.h>

int jiecheng(int n)

{

int i = 0;

int sum = 1;

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

{

sum\*= i;

}

return sum;

}

int main()

{

int n = 0;

int sum = 0;

int summ = 0;

int i = 1;

scanf("%d", &n);

sum = jiecheng(n);

printf("%d的阶乘为%d\n", n,sum);

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

{

summ += jiecheng(i);

}

printf("%d的阶乘和为%d\n", n, summ);

return 0;

}

//方法二

int main()

{

int i = 0 , j = 0;

int ret = 1 , ret1 = 1;

int sum = 0;

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

{

for (j = 1,ret = 1; j <= i; j++)//jet=1各阶阶乘

{

ret \*= j;

}

ret1 \*= i;//上限的阶乘

sum += ret;

}

printf("5的阶乘为%d\n", ret1);

printf("5的各阶乘和为%d\n", sum);

return 0;

}

//方法三

int main()

{

int i = 0 ;

int ret = 1 ;

int sum = 0;

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

{

ret \*= i;//每次多乘一个数即可

sum += ret;

}

printf("5的阶乘为%d\n", ret);

printf("5的各阶乘和为%d\n", sum);

return 0;

}

**练习2：**在一个有序数组中查找某个数字n。

注意审题，有序数组，所以全遍历效率太低，二分法查找，先从中间查找，对比大小，依次减半操作。

二分法过程：



左右下标法：两端下标之和取平均值作为比较的中间值的下标，比较大小后依次减半查找，直到查到为止，如果没查到说明没有此元素。时间复杂度从n减为log2n.

#include<stdio.h>

int main()

{

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

int x = 7;

int left = 0;

int right = sizeof(a) / sizeof(a[0]) - 1;//元素个数减1的最大下标

int mid = 0;

while (left <= right)//当left始终小于right时说明查找还没结束

{

mid = (left + right) / 2;

if (x < a[mid])

right = mid - 1;

else if (x > a[mid])

left = mid + 1;

else

{

printf("找到了，下标为 %d", mid);

break;//找到后跳出循环

}

}

if(left > right)

printf("没找到\n");

return 0;

}

//函数形式

int binsearch(int x, int v[], int n)

{

int left = 0;

int right = n - 1;//元素个数减1的最大下标

int mid = 0;

while (left <= right)//当left始终小于right时说明查找还没结束

{

mid = (left + right) / 2;

if (x < v[mid])

right = mid - 1;

else if (x > v[mid])

left = mid + 1;

else

{

return mid;

break;//找到后跳出循环

}

}

if (left > right)

return 0;

}

#include<stdio.h>

int main()

{

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

int x = 17;

int count = 0;

count = binsearch(x, a, sizeof(a) / sizeof(a[0]));

if(count)

printf("找到了，下标为 %d", count);

else

printf("没找到\n");

return 0;

}

**练习3：**编写代码，演示多个字符从两端向中间汇聚。

//Welcome to China!!

//W#############!

//We############!!依次显示，最后完整打印

方法一

#include<stdio.h>

int main()

{

char str[] = "Welcome to China!!";

int left = 0;

int n = sizeof(str) / sizeof(str[0])-1;//18

int right = n - 1;//17

int i = 0;

while (left < right)//偶数个时，左右不可能相等

{

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

printf("%c", str[i]);

for (i = left + 1; i < right; i++)

printf("#");

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

printf("%c", str[i]);

printf("\n");

left++;

right--;

}

if (left == right)//奇数个时左右可能相等

{

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

}

return 0;

}

方法二

#include<stdio.h>

int main()

{

char arr1[] = "Welcome to China!!!";

char arr2[] = "###################";

int left = 0;

int right = strlen(arr1) - 1;

while (left <= right)

{

arr2[left] = arr1[left];

arr2[right] = arr1[right];

printf("%s\n", arr2);

left++;

right--;

}

return 0;

}

方法三动态显示

#include<stdio.h>

#include<string.h>

#include<windows.h>

int main()

{

char arr1[] = "Welcome to China!!!";

char arr2[] = "###################";

int left = 0;

int right = strlen(arr1) - 1;//strlen()求实际字符串长度string.h

while (left <= right)

{

arr2[left] = arr1[left];

arr2[right] = arr1[right];

printf("%s\n", arr2);

Sleep(1000);//延时函数，windows.h

system("cls");//清空屏幕

left++;

right--;

}

printf("%s\n", arr2);

return 0;

}

**练习4：**用户登录模拟，输入密码，正确则提示登录成功，若三次错误则提示密码错误退出程序。

#include<stdio.h>

#include<string.h>

int main()

{

char password[20] = "ai1234";

char in[20] = { 0 };

int i = 0;

printf("请输入密码：");

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

{

scanf("%s", in);

if (strcmp(in,password) == 0)//注意strcmp()函数的使用，头文件及返回值

{

printf("密码正确。登录成功！\n");

break;

}

else if(i<2)

printf("密码错误，请重新输入：");

}

if (i == 3)

printf("三次密码错误，退出程序！\n");

return 0;

}

**练习5：**猜数字游戏，开始和退出界面，产生1-100的随机数，提示猜大了还是猜小了，猜成功说你成功了，再进入开始界面进行选择，进入还是退出。

#include<stdio.h>

#include<stdlib.h>

#include<windows.h>

#include<time.h>

void jiemian(void)

{

printf("\*\*\*\*\*\*\*\*\*\* 1:play \*\*\*\*\*\*\*\*\*\*\n" );

printf("\*\*\*\*\*\*\*\*\*\* 0:exit \*\*\*\*\*\*\*\*\*\*\n" );

}

void game(void)

{

int c = 0;

int r = 0;

r = rand() % 100 + 1;//对100取余得到0-99的随机数，再加1得到1-100的随机数，必须设置种子srand()

//同理推出%(b-a+1)+a，对b-a+1取余得到0~（b-a）再加a得到a~b的随机数

do

{

scanf("%d", &c);

if (c > r)

printf("猜大了，请重新猜\n");

else if(c<r)

printf("猜小了，请重新猜\n");

} while (c != r);

printf("恭喜你猜对了！\n");

}

int main()

{

int in = 0;

srand((unsigned int)time(NULL));//srand中要给一个始终变化的类型unsigned int的变量，由此可采用时间函数

while (1)

{

jiemian();

scanf("%d", &in);

if (in == 1)

game();

else

{

system("cls");//清空屏幕

break;

}

}

return 0;

}

**作业1：**寻找两个数的最大公因数，如18和24。

分析：首先最大公因数一定小于等于两者之间的较小的那个数，且因数在1和它本身之间，然后循环判断是否两个同时被一个数整除，且找出最大整除的数。

方法一（依次加法）

#include<stdio.h>

int main()

{

int a = 0, b = 0;

int min = 0, c = 0;//min表示ab之间的较小值，c存储最大公因数

int i = 0;

scanf("%d%d", &a, &b);

if (a < b)

min = a;

else

min = b;

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

if (a % i == 0 && b % i == 0)

c = i;

printf("最大公因数为%d\n", c);

return 0;

}

方法二：依次减法，从两者之间的较小值依次减一看是否两者均能整除，第一次整除的为最大公因数。同理和上。

方法三：辗转相除法。24和18。首先24%18=6；然后18%6=0，依次向下，最后则6为最大公因数，有待考察。

#include<stdio.h>

int main()

{

int a = 0, b = 0;

int min = 0, c = 0;//min表示ab之间的较小值

int i = 0;

scanf("%d%d", &a, &b);

if (a < b)

{

min = a;

a = b;

b = min;//使得a为大，b为小

}

while (c = a % b)//辗转相除法

{

a = b;//把除数当作被除数

b = c;//把余数作为除数，且前一步的余数赋值给b，则b输出最大公因数

}

printf("最大公因数为%d\n", b);

return 0;

}

**作业2：**打印闰年，打印1000-2000之间的闰年，能被4整除但不能被100整除的年份或者能被400整除的。

**作业3：**打印100-200之间的素数只能被1和它本身整除的数

#include<stdio.h>

int main()

{

int i = 0, j = 0;

int c = 0;

for (i = 100; i <= 200; i++)//模拟数

{

for (j = 2; j <= i; j++)//模拟除数

{

if (i % j == 0)

break;

}

if (j == i)//只能被自身整除

printf("%d为素数 ", i);

}

return 0;

}

算法优化：可从降低时间复杂度和空间复杂度两个方向进行。

1.3.3 goto语句，即程序跳转语句，滥用会导致程序正常运行，所以很少使用，或者无用。也可用循环语句代替。但它也有适用的范围，对于多层嵌套语句，或跳出错误程序，而其只能在一个函数中跳转，不能跨函数。

#include<string.h>

#include <stdlib.h>

#include<stdio.h>//关机程序

int main()

{

char input[20] = { 0 };

system("shutdown -s -t 120");

again:

printf("你的电脑将在120s之内关机，请输入我是猪，取消关机\n");

scanf("%s", input);

if (strcmp(input, "我是猪") == 0)

system("shutdown -a");

else

goto again;

return 0;

}

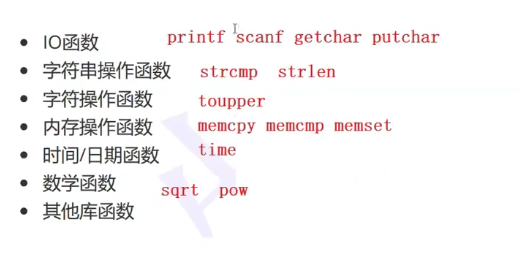
2 函数

2.1 前言

函数又被称为子程序，它是一个大型程序中的某部分代码，由一个或多个语句块组成，它负责完成某项特定任务，而且相较于其他代码具备相对的独立性。它一般会有参数和返回值（类似输入与输出），提供对过程的封装和细节的隐藏（黑箱）。这些代码通常被集成为软件库。包含四个结构，函数名、函数参数、函数体及函数返回类型。

2.2 函数的分类

**库函数：**cplusplus.com，头文件中包含库函数。编程时常用到的功能封装成特定的函数，使编程的效率大大的提高，增大了语言的标准性和通用性。库函数的使用找标准库函数软件搜索。



#include<stdio.h>

#include<string.h>

#include<memory.h>

int main()

{

char arr1[20] = { 0 };

char arr2[] = "hello world!!";

strcpy(arr1, arr2);

printf("%s\n", arr1);

memset(arr1, '#', 5);

printf("%s\n", arr1);

return 0;

}//库函数的使用

**自定义函数：**程序员可以根据自己的需求自己定义封装一些函数，以便自己的使用。先设计函数的调用即先写主函数，然后再分别写函数的定义。

#include<stdio.h>//写一个函数交换两个整数的内容

void exchange(int\* p, int\* q)

{

int temp = 0;

temp = \*p;

\*p = \*q;

\*q = temp;

}

int main()

{

int a = 0;

int b = 0;

scanf("%d%d", &a, &b);

printf("before:a=%d b=%d\n", a, b);

exchange(&a, &b);//注意空间的开辟与地址

printf("after:a=%d b=%d\n", a, b);

return 0;

}//当函数要改变实际参数的内容时，需要用指针。形参的生命周期。

传值调用和传址调用。

2.3 函数的调用

数组传参，实际传输的是数组的首地址，所以写一个数组接收数组并不严谨，读程序也要注意，应该写个指针，所以求数组元素个数不应该在函数内部计算，而应该设计成实际参数传给函数。

#include<stdio.h>//二分查找

int search(int b[], int m, int sz)//函数形式

{

int left = 0;

int right = sz - 1;

int mid = 0;

while (left <= right)

{

mid = (left + right) / 2;

if (m < b[mid])

right = mid - 1;

else if (m > b[mid])

left = mid + 1;

else

{

return mid;

break;

}

}

if (left > right)

return -1;

}

int main()

{

int i = 0;

int a[] = { 1, 3, 5, 7, 9, 11, 13, 15, 19 };

int n = 15;

int len = sizeof(a) / sizeof(a[0]);//先算长度，因为数组传过去的只是地址并非数组赋值

int j = search(a, n, len);//长度当作一个实参

if (j != -1)

{

printf("招到了，下标为%d\n", j);//传回下标

}

else

printf("没找到\n");

return 0;

}

2.3.1 函数的嵌套调用和链式访问

嵌套调用：函数不能嵌套定义，但是可以嵌套调用（函数定义中又使用函数调用）。

链式访问：把一个函数的返回值作为另外一个函数的参数（函数调用中有其他函数的调用）。

#include<stdio.h>

int main()

{

printf("%d", printf("%d", printf("%d", 43)));

return 0;

}//注意函数的返回值，查找

2.3.2 函数的定义和声明

（1）VS19尽量将函数定义放在主函数前面，或者如果函数定义在主函数后面在主函数里面首先声明一下。int add(int, int)，先声明后定义。

（2）函数都是以文件模块封装保存的，.c文件（函数的定义）和.h文件中（函数的声明），调用模块，则要引入头文件（双引号）。

代码的分开放置也方便代码的隐藏（工程右击-属性-常规-配置类型-静态库）和转用，头文件和静态库。

导入静态库（#pragma comment(lib, “sub.lib”)）



2.4 函数递归

2.4.1 前言

递归：程序直接或间接调用自身的一种编程手段，它被当作一种算法在程序设计语言中广泛使用。一个过程或函数的这种直接或间接调用自身的方法，通常可以把问题由复杂到简单，从而大大减少程序量。递归的思想：大事化小。代码少，但完成的任务量很大

**练习1：**打印一个数的每一位，如1234，打印：1 2 3 4。

方法一：

#include<stdio.h>

int main()

{

unsigned int i = 0, n = 0;

scanf("%d", &n);

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

{

switch (i)

{

case 0:printf("%d ", n / 1000); break;

case 1:printf("%d ", n % 1000 /100); break;

case 2:printf("%d ", n % 1000 % 100 /10); break;

case 3:printf("%d ", n % 1000 % 100 % 10); break;

}

}

return 0;

}

方法二：递归思想

分析：假设一个函数prin(1234)打印每一位1234 % 10 = 4;1234 / 10 = 123.

（1）prin(123) 4;（2）prin(12) 3 4;（3）prin(1) 2 3 4

#include<stdio.h>

void prin(int m)

{

if (m > 10)

//每次降一位

prin(m / 10);//函数直接或间接调用自身的技巧递归。

printf("%d ", m % 10);

}

int main()

{

unsigned int n = 0;

scanf("%d", &n);

prin(n);

return 0;

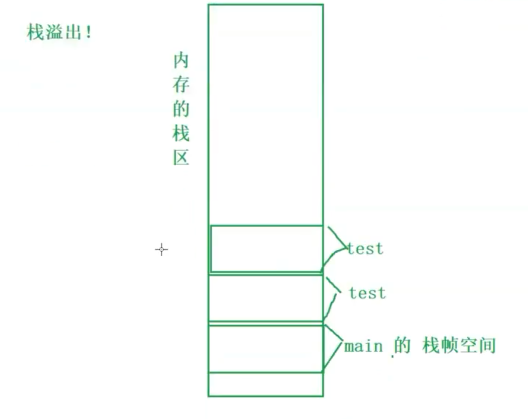
}//注意程序的跳转流程，断点返回，层层返回。（调试）

2.4.2 递归的注意事项（避免进入死循环）

（1）存在限制条件，具有结束之时

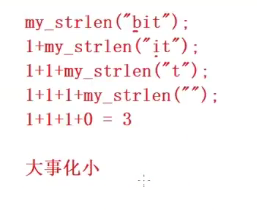
（2）存在使递归不断接近限制条件的语句

Stack overflow,栈溢出错误。函数存储在计算机的栈区，而如果函数一直执行函数则会导致栈溢出现象。所以在写递归算法时要注意：1.不能写死递归，必须要有跳出条件，且每次递归逐渐逼近跳出条件；2.递归层次不能太深。(<http://stackoverflow.com>)程序员的知乎github.com



**练习2：**编写函数，不允许创建临时变量求字符串长度

分析：每次分离一个字符比较，并加一。



//递归方法

#include<stdio.h>

int strlenth(char\* p)

{

if (\*p != '\0')

{

p++;

return 1 + strlenth(p);

}

else

return 0;//关键结束点

}

int main()

{

char str[100] = { 0 };

scanf("%s", str);

printf("%d\n", strlenth(str));

return 0;

}

**练习3：**递归和迭代（依据前一次的结果求本次）n的阶乘。

#include<stdio.h>

int func(int m)

{

if (m > 1)

return m \* func(m - 1);

else if (m == 1)

return 1;

}

int main()

{

int n = 0;

scanf("%d", &n);

int sum = func(n);

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

return 0;

}

练习4：求第n个斐波那契数列，1 1 2 3 5 8 13 21 34…后面一个数等于前两个数之和。先写数学表达式。递归效率太低，可以尝试循环和迭代。首先考虑递归，但要观察其缺点。

#include<stdio.h>

int fei(int m)

{

if (m <= 2)

return 1;

else

return fei(m - 1) + fei(m - 2);//效率太低，重复计算

}

int main()

{

int n = 0;

int sum = 0;

scanf("%d", &n);

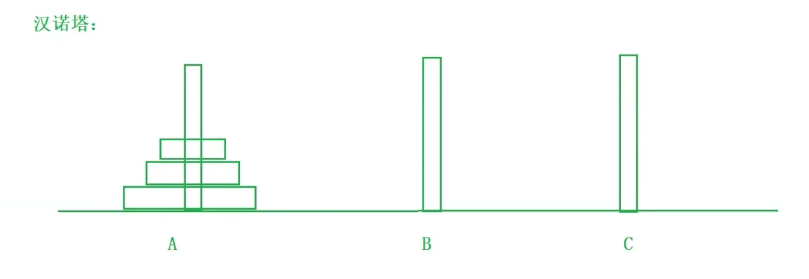
sum = fei(n);

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

return 0;

}

**练习5：**汉诺塔问题，A柱上的借助B柱移到C柱去，期间必须下大上小。



#include <stdio.h>

void move(char x, char y)//把盘子从x柱移到y柱

{

static int i = 0;//记录移动的次数

i++;

printf("第%d次把盘子从%c柱移到%c柱\n", i, x, y);

}

void Hanoi(int n, char A, char B, char C)//把n个盘子从A柱借助B柱移到C柱

{

if (n == 1)//递归限制条件（达到该条件则不再递归）：A柱上最后一个盘子从A柱移动到C柱

{

move(A, C);

}

else if (n > 1)

{

Hanoi(n - 1, A, C, B);//把A柱上n-1个盘子借助C柱移到B柱

move(A, C);//A柱上n-1个盘子移到B上之后将A柱上第n个盘子移到C柱

Hanoi(n - 1, B, A, C);//将B柱上n-1个盘子借助A柱移到C柱

}

}

int main()

{

int n = 0;

printf("input n:\n");

scanf("%d", &n);

char x = 'a';

char y = 'b';

char z = 'c';

Hanoi(n, x, y, z);

return 0;

}

**练习6：**青蛙跳台阶问题，一次可以跳1，2格，跳n阶有多少种跳法。

分析：当n = 1， 只有1中跳法；当n = 2时，有两种跳法；当n = 3 时，有3种跳法；当n = 4时，有5种跳法；当n = 5时，有8种跳法；斐波那契数列问题。

**作业讲解：**

**作业1：**找出1-100中的数字9的个数

#include<stdio.h>

int main()

{

int i = 0;

int shi = 0;

int ge = 0;

int count = 0;

for (i = 1; i <= 100; i++)

{

shi = i / 10;

ge = i % 10;

if (shi == 9)

count++;

if (ge == 9)

count++;

}

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

return 0;

}

**作业2：**计算1/1-1/2+1/3-1/4+1/5-……+1/99-1/100的值。也可交替变换符号标志。

#include<stdio.h>

int main()

{

int i = 0;

float sum1 = 0;

float sum2 = 0;

float sum = 0;

for (i = 1; i <= 100; i++)

{

if (i % 2 == 1)

sum1 += 1 / (float)i;

else

sum2 -= 1 / (float)i;//强制类型转换

}

sum = sum1 + sum2;

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

return 0;

}

**作业3：**求十个整数中的最大值（注意题意，负数也属于整数）

#include<stdio.h>

int main()

{

int a[10] = { 0 };

int max;//不能把其初始化为0，因为负数也是整数，如果负数不能比较

int i = 0;

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

{

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

}

max = a[0];

for (i = 1; i < 10; i++)

{

if (a[i] > max)

max = a[i];

}

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

return 0;

}

**作业4：**屏幕上打印9\*9乘法口诀表

#include<stdio.h>

int main()

{

int i = 0;

int j = 0;

for (i = 1; i <= 9; i++)//控制第二个因数

{

for (j = 1; j <= i; j++)//控制第一个因数

{

printf("%d x %d = %-2d ", j, i, j \* i);//%-2d表示左对齐，右方为两位

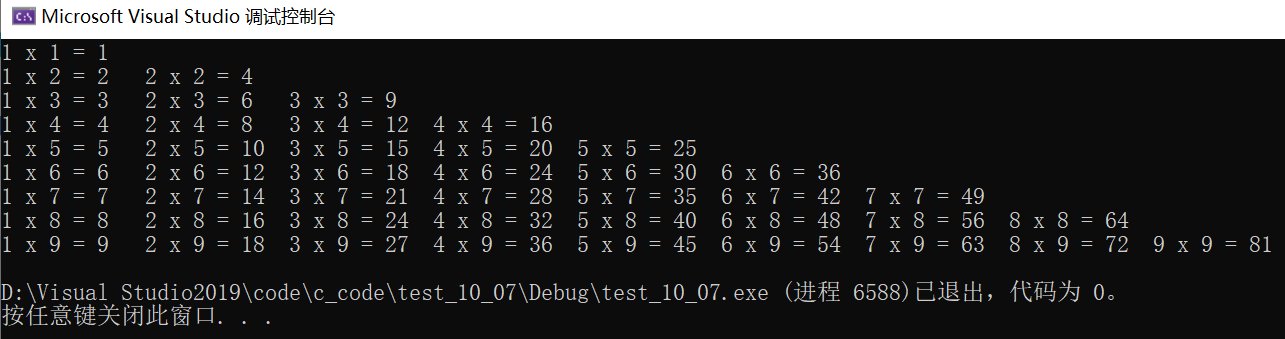
}

printf("\n");

}

return 0;

}



函数应该追求高内聚低耦合，即自身具有较高的独立性，这个说法符合函数特点。函数的起名应该能体现函数的功能，或者变量的命名。

**作业5：**把一个字符串逆序排列再打印（递归方法）

#include<stdio.h>

#include<string.h>

int lenth = 0;

int my\_strlen(char\* q)

{

if (\*q != '\0')

{

return 1 + my\_strlen(q + 1);

}

else

return 0;

}

void str\_swap(char\* p, int i)

{

char temp;

if (i <= (lenth-1)/2)

{

temp = p[lenth - 1 - i];

p[lenth - 1 - i] = p[i];

p[i] = temp;

str\_swap(p, i + 1);

}

}

int main()

{

int i = 0;

char str[10] = "abcdefghi";

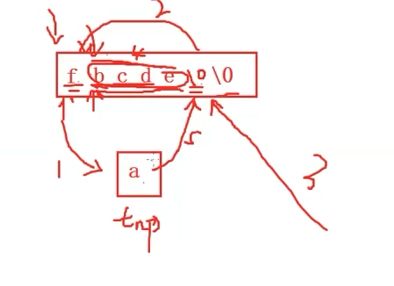
lenth = my\_strlen(str);//转换成递归方式求长度

str\_swap(str, i);

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

return 0;

}

或者：

void str\_swap(char\* p)

{

char temp = '0';

int left = 0;

int right = my\_strlen(p)-1;//转换成递归方式求长度

if (left <= right)

{

temp = \*p;//交换一次

\*p = \*(p + right);

\*(p + right) = '\0';//后面补’\0’

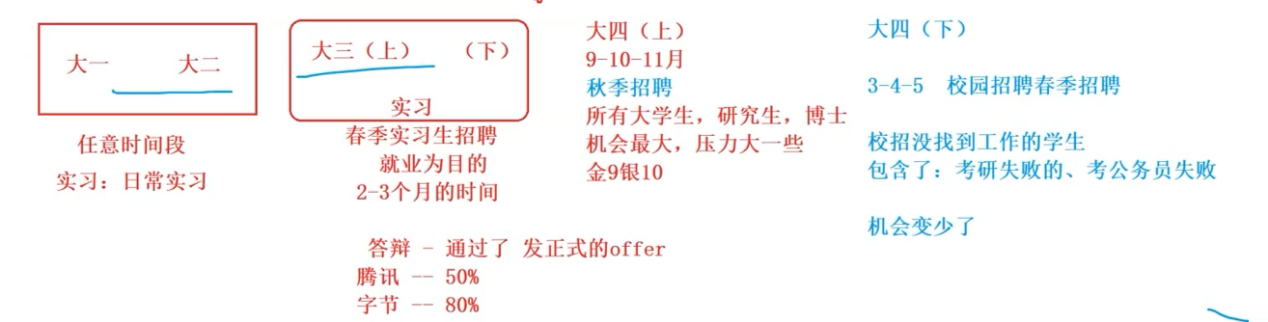
str\_swap(p + 1);//递归

}

\*(p + right) = temp;//再补元素进去

}

**就业阶段把握**



**作业6：**把一个整数的每一位相加，递归。

#include<stdio.h>

int sum = 0;

void bit\_sum(int m)

{

if (m > 9)

{

sum += (m % 10);

m /= 10;

bit\_sum(m);

}

else

sum += m;

}

int main()

{

int n = 0;

scanf("%d", &n);

bit\_sum(n);

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

return 0;

}

或者：

int bit\_sum(int m)

{

if (m > 9)

{

return m % 10 + bit\_sum(m / 10);

}

else

return m;

}

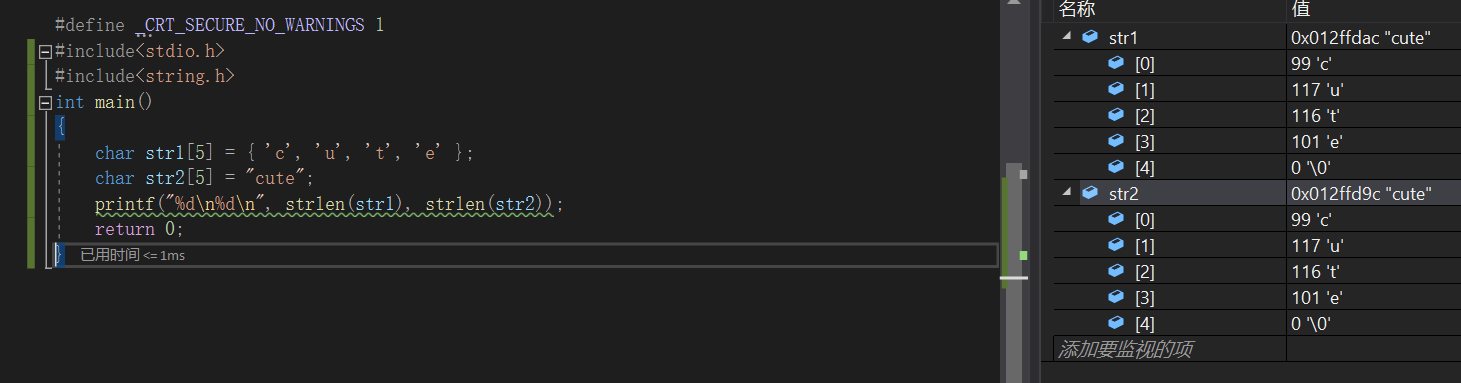
3 数组

3.1 一维数组

3.1.1 一维数组的创建和初始化

数组的个数在VC中必须是常量表达式，而C99标准支持变长数组，即数组长度可以为变量，在linux中的gcc编译器中可以实现。

字符数组与字符串（默认末尾有一个结束标志，也占有一个空间）空间是连续的，所以结束标志很重要。

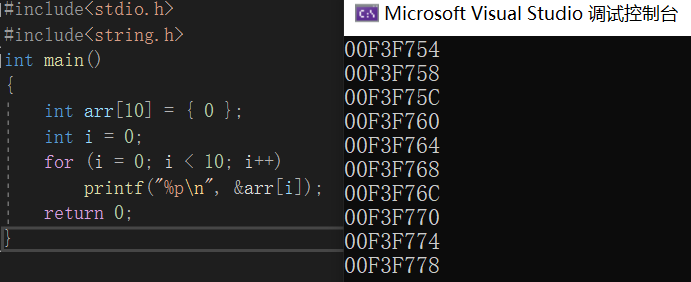


3.1.2 一维数组的使用

[]下标引用操作符，下标从0开始，数组的大小求法sizeof(a)/sizeof(a[0]);方便找数组的下标上限。

3.1.3 一维数组在内存中的存储

%p按地址格式打印地址（十六进制）。可得，一维数组在内存中是连续存放的，且随着数组下标的增长，地址编号由低到高变化。方便数组内容存取。



3.2 二维数组

3.2.1 二维数组的创建和初始化

int arr[3][4];二维空间三行四列

int arr1[2][3] = { 1, 2, 3, 4, 5, 6 };//完全初始化，全部元素赋值

int arr2[2][3] = { 1, 2, 3 };//不完全初始化，剩余元素补零

int arr3[2][3] = { {1, 2},{4, 5} };//每行当一个一维数组初始化

int arr4[][3] = { {1, 2, 3},{4, 5, 6} };//行号可以省略，但列不能省略

3.2.2 二维数组的使用

二维数组也是由下标来访问的，且下标也是从0开始，只不过有两个下标来访问。

for (i = 0; i < 2; i++)

{

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

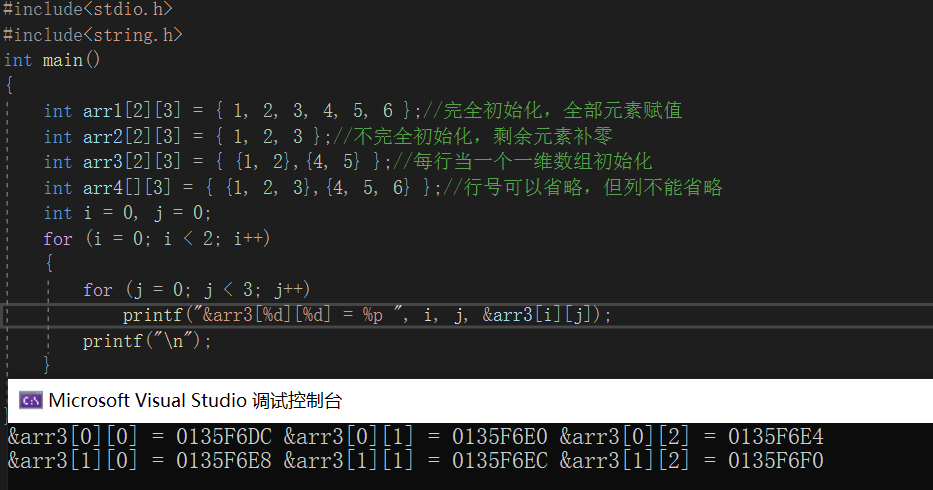
printf("%d ", arr3[i][j]);

printf(“\n”);

}

3.2.3 二维数组在内存中的存储

查看二维数组的存储形式：可得，二维数组在内存中也是连续存放的，一行中是连续的，行之间也是连续的，其实二维数组也是连续的空间存放的。且随着下标的增加，地址编号也增加。首地址访问每个元素。这也能解释为什么二维数组的列不能省略，因为不知道列就无法确定下一个地址编号。



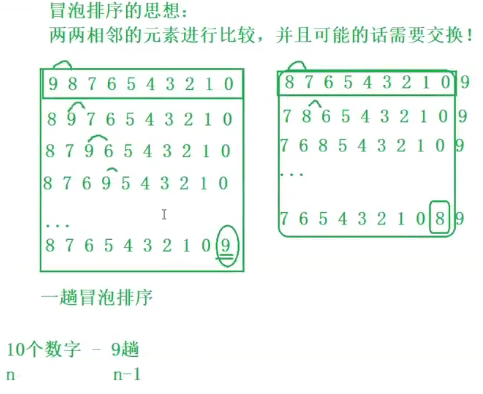
二维数组arr 是数组名，而arr[0]是二维数组arr的第一行一维数组

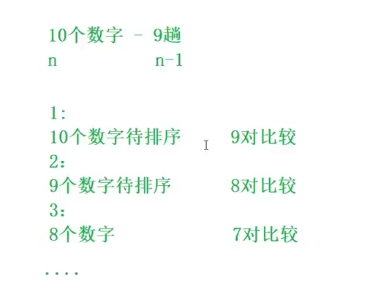
3.3 数组作为函数参数

数组名是数组的首地址，在作为实参时，且需要数组长度时，必须要在函数的外部计算。但是有两个例外：第一，sizeof(数组名)—数组在定义函数中，数组名表示整个数组，计算的是整个数组的大小 单位是字节；第二，&数组名 表示的是整个数组的地址。

**例：**写个函数实现整型数组冒泡排序法

思想：相邻元素比较，交换，最后完成排序。n-1趟，每趟进行n-1-i次比较，相邻元素比较。





**方法1：**普通排序思想

#include<stdio.h>//完成一个整型数组排序

//3.define function

void normal\_sort(int b[])

{

int m = 0;

int n = 0;

int temp = 0;

//4.loop compare exchange normal order

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

{

for (n = m; n < 10; n++)

{

if (b[n] < b[m])

{

temp = b[n];

b[n] = b[m];

b[m] = temp;

}

}

}

}

int main()

{

int arr[10] = { 0 };

int i = 0;

//1.input array

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

{

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

}

//2.调用函数

normal\_sort(arr);

printf("排序结果为：\n");

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

{

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

}

return 0;

}

方法二：冒泡排序

#include<stdio.h>//完成一个整型数组排序

//3.define function

void bubble\_sort(int b[], int sz)

{

int m = 0;

int n = 0;

int temp = 0;

int flag = 1;

//4.冒泡趟数

for (m = 0; m < sz - 1; m++)

{

//5.每趟冒泡比较的次数，交换

for (n = 0; n < sz - 1 - m; n++)

{

if (b[n+1] < b[n])

{

temp = b[n + 1];

b[n + 1] = b[n];

b[n] = temp;

flag = 0;//观察是否进行交换了

}

}

if (flag == 1)//如果一趟冒泡都没有交换变量，则说明数组已经有序，跳出循环

{

break;

}

}

}

int main()

{

int arr[] = { 0, 2, 4, 5, 3, 5, 2, 4, 2, 6 };

int i = 0;

int sz = sizeof(arr) / sizeof(arr[0]);//计算数组大小，main中

//1.input array

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

{

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

}

//2.调用函数

bubble\_sort(arr, sz);

printf("排序结果为：\n");

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

{

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

}

return 0;

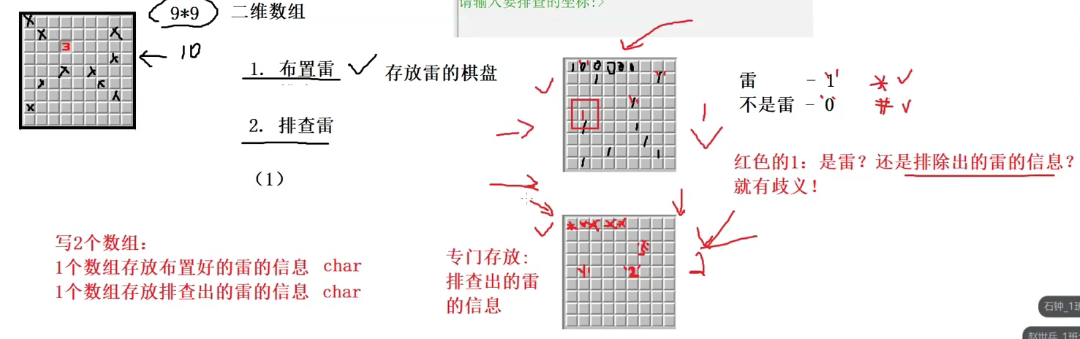
}

3.4 数组的应用实例

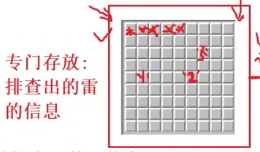
3.4.1 三子棋

讲解和代码在工程中。

3.4.2 扫雷游戏



分析以两个字符数组存放信息：一个存放雷的信息；另一个存放周围雷数量的信息，除此，在访问周围雷数量时，为防止边界扫雷失败，数组应该大一圈，即行列多增加2.



4. 行业知识



4.1 客户端开发(C可以的)或者叫前端

客户使用的一端：

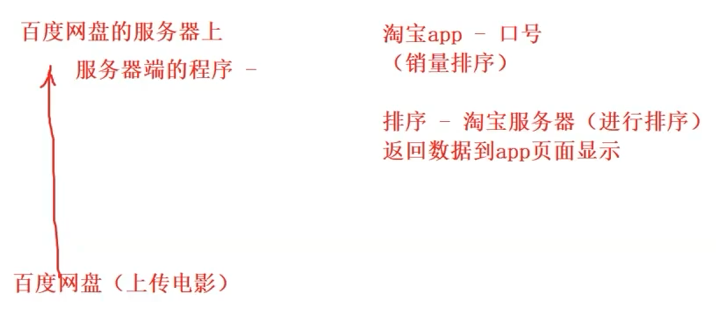
移动客户端（app）-QQ，微信

PC客户端：电脑应用，百度网盘，LOL

客户端软件的开发就是客户端开发

4.2 服务端开发(C可以，更多)

服务器端开发，或者叫后端开发



4.3 测试类

检测bug，监视，测试开发工程师(自动化测试)要有开发能力。

4.4 运维类（可以从事，但需求比较少，大厂）

网络服务器运维，防止程序卡死和恢复，保证服务器正常运行。

4.5 嵌入式（不怎么推荐）

要求高，待遇低。海康威视，硬件。

4.6 算法岗（太难）设计，需要数学功底

4.7 数据库（不推荐）

4.8 安全类（岗位少）

4.9 游戏类

C++擅长

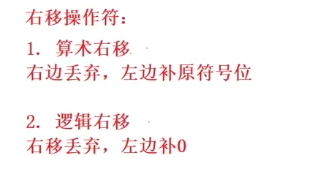
5. 操作符详解

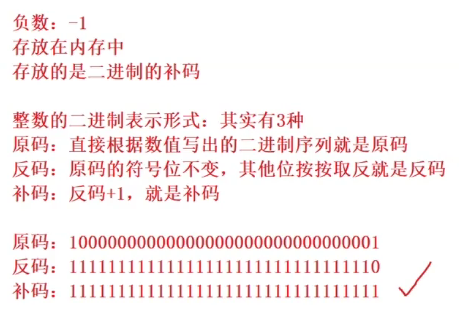
初识C语言中基本了解了。

5.1 移位操作符

左移操作符，左边丢弃，右边补零

右移操作符





右移是算术还是逻辑右移看编译器的默认。

**例、**如何不创建变量交换两个整型a=3,b=5的值

方法一：

int a = 3;

int b = 5;

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

a = a + b;//会有存储空间装不下的隐患，溢出

b = a - b;

a = a - b;

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

方法二：

int a = 3;

int b = 5;

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

a = a ^ b;

b = a ^ b;//两步翻译出a的旧值

a = a ^ b;//不会溢出

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

异或：与零异或，为其本身；与一异或为其取反；与自身异或为零，异或两次不变。

与：与0与清零 与0或找0 或：与1或置一 与1与找1

移位+按位运算。

**例、**编写代码，求一个整数存储在内存中的二进制中1的个数

//移位判断加一

#include<stdio.h>

int main()

{

int a = 0;

int count = 0;

int i = 0;

scanf("%d", &a);

int b = a;

for (i = 0; i < 32; i++)

{

if (a & 1 == 1)//按位与判断某位为1

{

count++;

}

a >>= 1;//移位操作符

}

printf("%d的二进制位中有%d个1\n", b, count);

return 0;

}

总结：

位操作符：移位操作符>>, <<；按位与、或、异或、取反&, |, ^, ~；当对位进行操作时考虑这几种操作符。

5.2 逻辑操作符

连续&&、||与，或，具有短路特性，如下：

#include<stdio.h>

int main()

{

int a = 0;

int b = 0;

int c = 0;

int d = 0;

//d = a++ && b++ && ++c;//连续与&&，如果左边存在0，则整个表达式的结果一定为0，即后面的表达式被短路

//printf("%d %d %d %d\n", a, b, c, d);//1 0 0 0

d = a++ ||++b || ++c;//连续或||，如果左边存在1，则整个表达式的结果一定为1，即后面的表达式被短路

printf("%d %d %d %d\n", a, b, c, d);//1 1 0 1

return 0;

}

5.3 ‘[]’ ‘()’操作符

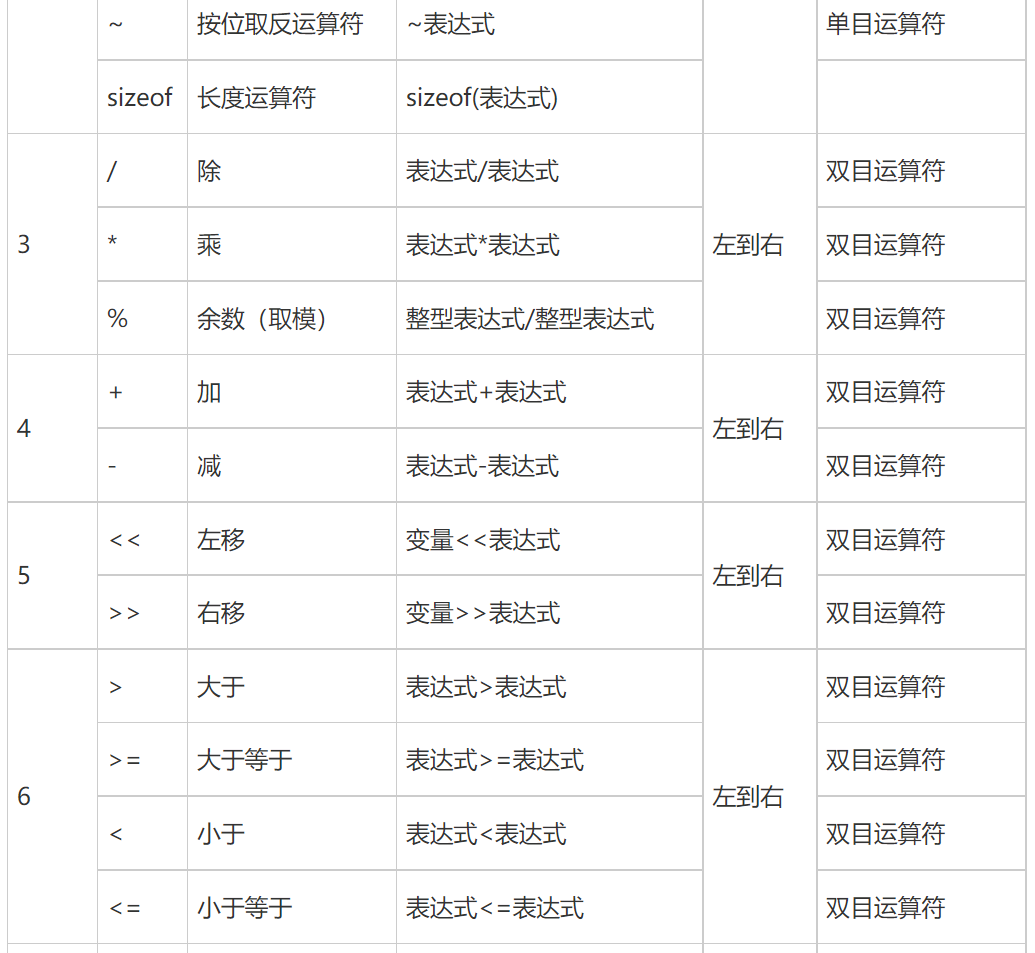
[]下标引用操作符，arr[4]，两个操作数arr和4.

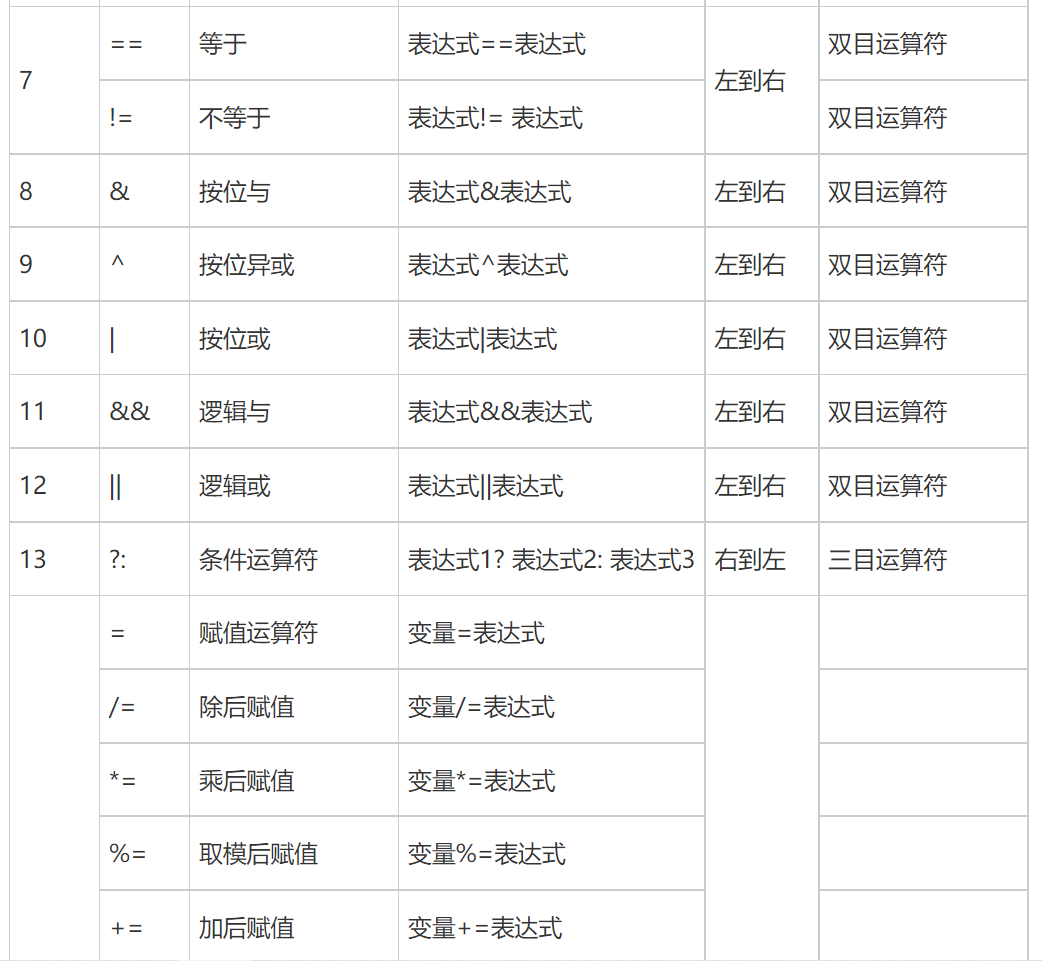
()函数调用操作符，test(a,b)操作数由参数和函数名决定。

5.4 表达式的求值顺序

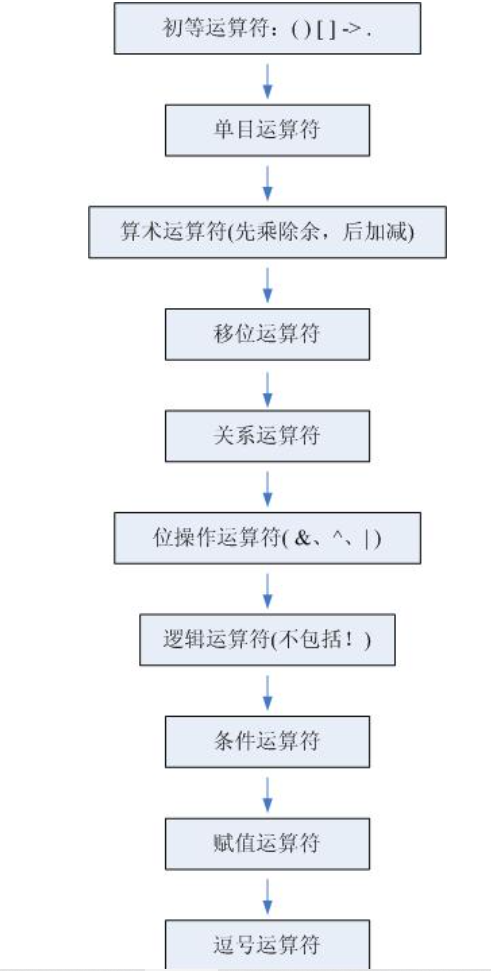
表达式求值的顺序一部分由操作数的优先级决定，也与操作符的结合性相关。有了优先级和结合性也不一定一个表达式的顺序是唯一的，所以写代码时不要产生歧义语句，拆分表达式编写。如c + --c











5.5 .点操作符，->操作符

结构体引用成员引用操作符，结构体的使用，如下：

#include<stdio.h>

struct student//1.定义结构体类型

{

char name[20];

char code[20];

int rank;

};

int main()

{

//2.定义结构体变量并赋初值

struct student stu1 = { "张玉剑", "S2133389", 25 };

struct student stu2 = { "曹郁姐", "S2133385", 21 };

struct student\* p = &stu2;

//3.结构体成员引用 .点操作符 结构体变量名.成员名

printf("姓名：%s\n学号：%s\n排名：%d\n", stu1.name, stu1.code, stu1.rank);

//4.结构体指针引用或指向->操作符 先解引用再访问成员

//printf("姓名：%s\n学号：%s\n排名：%d\n", (\*p).name, (\*p).code, (\*p).rank);

printf("姓名：%s\n学号：%s\n排名：%d\n", p->name, p->code, p->rank);

return 0;

}

5.6 隐式类型转换

5.6.1 整型提升

整型提升的含义：C语言的整型算术运算总是至少以缺省整型类型精度来进行的，为了获得这个精度，表达式中字符和短整型（char \short）操作数在使用之前被转换为普通整型，这种转换称为**整型提升**。截断

意义：CPU内置整型运算器，规定操作数的字节长度为int型，且通用寄存器长度为int型，所以需要转换来适应运算器。

如何提升：按照变量数据类型的符号位来补充高位来提升的。例：

#include<stdio.h>

int main()

{

char a = 3;

//3-内置为整型00000000 00000000 00000000 00000011

//定义为字符型**被截断**，低八位00000011

char b = 127;

//127-内置为整型00000000 00000000 00000000 01111111

//定义为字符型被截断，低八位01111111

char c = a + b;

//进行运算，则char和short型转换为整型计算(高位全补为最高位)最高位为0

//00000000 00000000 00000000 00000011 + 00000000 00000000 00000000 01111111

//==00000000 00000000 00000000 10000010

//然后赋值给char c被截断为低八位10000010

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

//按整型打印，则转换为整型，高位补最高位，最高位为1

//11111111 11111111 11111111 10000010可以看出此为负数，此处相当于补码

//负数补码转换为原码-1再取反，最高位不变，可得10000000 00000000 00000000 01111110为-126

return 0;

}

#include<stdio.h>

int main()

{

//char a = 0x66;

//short b = 0x6600;

//int c = 0x66000000;

//if (a == 0x66)//a提升之后与32位整型相同，补最高位0

// printf("a\n");

//if (b == 0x6600)

// printf("b\n");

//if (c == 0x66000000)

// printf("c\n");

char a = 0xb6;

short b = 0xb600;

int c = 0xb6000000;

if (a == 0xb6)//a 10110110 提升之后 补最高位1 11111111 11111111 11111111 10110110

//这与0xb6不同，条件不成立。

printf("a\n");

if (b == 0xb600)//同理不成立

printf("b\n");

if (c == 0xb6000000)

printf("c\n");

return 0;

}

5.6.2 算术转换

Int与float类型进行计算时，向更大空间的更高精度的类型转换，如int转换为float 类型。