程序设计II：期末复习

C语言回顾

字/字节/Bit等单位

B与bit

数据存储是以“字节”（Byte）为单位，数据传输大多是以“位”（bit，又名“比特”）为单位，一个位就代表一个0或1（即二进制），每8个位（bit，简写为b）组成一个字节（Byte，简写为B），是最小一级的信息单位。

字（Word）

在计算机中，一串数码作为一个整体来处理或运算的，称为一个计算机字，简称字。字通常分为若干个字节（每个字节一般是8位）。在存储器中，通常每个单元存储一个字。因此每个字都是可以寻址的。字的长度用位数来表示。

字长

计算机的每个字所包含的位数称为字长，计算的字长是指它一次可处理的二进制数字的数目。一般地，大型计算机的字长为32-64位，小型计算机为12-32位，而微型计算机为4-16位。字长是衡量计算机性能的一个重要因素。

基本I/O函数回顾

scanf()返回值

>0： 表示正确输入参数的个数；

0： 没有项被赋值；

EOF：第一个尝试输入的字符是EOF(结束) （对OJ上某些题，返回值为EOF可以用来判断输入数据已经全部读完）

逐字符从stdin读取/向stdout输出字符

```
#include<stdio.h>
int getchar(void);
int putchar(int);
```

一般来讲，它比我们常用的scanf要快。

在未知数据组数时如何依次读取

```
char str[10];
int n;
while (scanf("%s%d", str, &n) == 2){
    solve();
}
```

或者：

```
char ch;
while ((ch = getchar()) != EOF){
    ......
}
```

对于后一种使用getchar()的代码，小心每行的回车符。

printf()返回值

成功打印的字符数；返回负值为出错。

格式控制符

%e　以科学计数法格式输出数值

%x　以十六进制读入或输出 int 变量

%I64d %lld　读入或输出 __int64 / long long 变量(64位整数)

%p　输出指针地址值

%.5lf　输出浮点数，精确到小数点后5位

读取一行

C：gets()（C新标准中被删除，但是上机时还可以用），fgets()，gets_s()等

C++：当 cin 读取数据时，它会传递并忽略任何前导白色空格字符（空格、制表符或换行符）。一旦它接触到第一个非空格字符即开始阅读，当它读取到下一个空白字符时，它将停止读取。可以输入 "Mark" 或 "Twain"，但不能输入 "Mark Twain"，因为 cin 不能输入包含嵌入空格的字符串。getline 函数如下所示：

```
    getline(cin, inputLine);
```

其中 cin 是正在读取的输入流，而 inputLine 是接收输入字符串的 string 变量的名称。

```
    cin.getline(szLine,Buffer_Size)
```

或者用cin.getline 读取一行，第一个参数是缓冲区地址；第二个参数是缓冲区大小，为了防止越界用的。缓冲区不够大，就自动截断。它会自动往缓冲区末尾添加 '\0'。

sscanf()函数和 sprintf()函数（socket项目中用到了）

```
int sscanf(const char * buffer, const char * format[, address, ...]);
```

和scanf的区别在于，它是从buffer里读取数据

```
int sprintf(char *buffer, const char *format[, argument, ...]);
```

和printf的区别在于，它是往buffer里输出数据

怎么判断读到空行?

`char s[100];`

可以用 `gets(s)` 一次读取一行，然后再用

`sscanf` 从 `s` 中分出英文和外文

位运算

如何判断一个int型变量n的第7位（从右往左，从0开始数）是否是1？

只需看表达式 `n & 0x80` 的值是否等于0x80即可。

或者 `if(n & (1<<6))>0)` ，即将1左移6位得到0x80。

如果需要将int型变量n的低8位全置成1，而其余位不变?

按位或运算通常用来将某变量中的某些位置1或保留某些位不变。

`n |= 0xff;`

进行简单的加密解密?

异或运算的特点是：如果 a^b=c，那么就有c^b = a以及c^a=b。此规律可以用来进行最简单的加密和解密。

左移运算

左移1位，就等于是乘以2，左移n位，就等于是乘以2n。

左移操作比乘法操作快得多。

左移的右操作数指明了要左移的位数。

左移时，高位丢弃，低位补0。

左移运算符不会改变左操作数的值。

右移运算

对于有符号数，如long,int,short,char类型变量，在右移时，符号位（即最高位）将一起移动，并且大多数C/C++编译器规定，如果原符号位为1，则右移时高位就补充1，原符号位为0，则右移时高位就补充0。

对于无符号数，如 `unsigned long`，`unsigned int`，`unsigned short`，`unsigned char` 类型的变量，则右移时，高位总是补0。

右移n位，就相当于左操作数除以2n，并且将结果往小里取整。

函数指针

函数指针定义

程序运行期间，每个函数都会占用一段连续的内存空间。而函数名就是该函数所占内存区域的起始地址(也称“入口地址”)。

我们可以将函数的入口地址赋给一个指针变量，使该指针变量指向该函数。然后通过指针变量就可以调用这个函数。

函数指针定义的一般形式为：

`类型名 (* 指针变量名)(参数类型1, 参数类型2,…);`

其中 `类型名` 表示被指函数的返回值的类型。 `(参数类型1, 参数类型2,……)` 中则依次列出了被指函数的所有参数的类型。

`int (*pf)(int ,char);` 表示pf是一个函数指针，它所指向的函数，返回值类型应是int，该函数应有两个参数，第一个是int类型，第二个是char类型。

可以用一个原型匹配的函数的名字给一个函数指针赋值。

```
#include <stdio.h>
void PrintMin(int a,int b)
{
    ...
}
int main(void){
    void (* pf)(int ,int);
    int x = 4, y = 5;
    pf = PrintMin;
    pf(x,y);
    return 0;
}
```

快速排序库函数qsort()

原型：

```
void qsort(void *base, int nelem, unsigned int width, int ( * pfCompare)( const void *, const void *));
```

base是待排序数组的起始地址，

nelem（number of element）是待排序数组的元素个数，

width是待排序数组的每个元素的大小（以字节为单位）

pfCompare是一个函数指针，它指向一个“比较函数”。该比较函数应是返回值为int,有两个参数为const void * 的函数

qsort函数在执行期间，会通过pfCompare指针调用 “比较函数”，调用时将要比较的两个元素的地址传给“比较函数”，然后根据“比较函数”返回值判断两个元素哪个更应该排在前面。这个“比较函数”不是C/C++的库函数，而是由使用qsort的程序员编写的。在调用qsort时，将“比较函数”的名字作为实参传递给pfCompare。程序员当然清楚该按什么规则决定哪个元素应该在前，哪个元素应该在后，这个规则就体现在“比较函数”中。

qsort函数的用法规定，“比较函数”的原型应是：

```
int 函数名(const void * elem1, const void * elem2);
```

该函数的两个参数，elem1和elem2，指向待比较的两个元素。

也就是说， * elem1和 * elem2就是待比较的两个元素。该函数必须具有以下行为：

1) 如果 * elem1应该排在 * elem2前面，则函数返回值是负整数（任何负整数都行）。

2) 如果 * elem1和* elem2哪个排在前面都行，那么函数返回0

3) 如果 * elem1应该排在 * elem2后面，则函数返回值是正整数（任何正整数都行）。

实例：

```
#include <stdio.h>
#include <stdlib.h>
#define NUM 5

int MyCompare( const void * elem1, const void * elem2 )
{
    unsigned int * p1, * p2;
    p1 = (unsigned int *) elem1;
    p2 = (unsigned int *) elem2;
    return  (* p1 % 10)  - (* p2 % 10 );
}

int main(void)
{
    unsigned int an[NUM] = { 8,123,11,10,4 };
    qsort( an,NUM,sizeof(unsigned int),MyCompare);
    for( int i = 0;i < NUM; i ++ )
        printf("%d ",an[i]);
    return 0;
}
```

动态内存分配

`T* Ptr = new T;`

T是任意类型名，P是类型为T * 的指针。这样的语句，会动态分配出一片大小为 sizeof(T)字节的内存空间，并且将该内存空间的起始地址赋值给P。

`T* P = new T[N];`

T是任意类型名，P是类型为T * 的指针，N代表“元素个数”，它可以是任何值为正整数的表达式，表达式里可以包含变量、函数调用。这样的语句动态分配出 N × sizeof(T)个字节的内存空间，这片空间的**起始地址**被赋值给P。

如果要求分配的空间太大，操作系统找不到足够的内存来满足，那么动态内存分配就会失败。保险做法是在进行较大的动态内存分配时，要判断一下分配是否成功。

`delete ptr;`

该指针必须是指向动态分配的内存空间的，否则运行时很可能会出错。

如果是用new动态分配了一个数组，那么，释放该数组的时候，应以如下形式使用 delete 运算符：

`delete [] ptr;`

命令行参数

将用户在DOS窗口输入可执行文件名的方式启动程序时，跟在可执行文件名后面的那些字符串，称为“命令行参数”。命令行参数可以有多个，以空格分隔。

比如，在Dos窗口敲：

```
copy file1.txt file2.txt
```

“copy”, “file1.txt”, “file2.txt”就是命令行参数。

如何在程序中获得命令行参数呢?

```
int main(int argc, char * argv[]){……}
```

参数argc就代表启动程序时，命令行参数的个数。C/C++语言规定，可执行程序程序本身的文件名，也算一个命令行参数，因此，argc的值至少是1。

参数argv是一个数组，其中的每个元素都是一个char* 类型的指针，该指针指向一个字符串，这个字符串里就存放着命令行参数。例如，**argv[0]指向的字符串就是第一个命令行参数，即可执行程序的文件名**，argv[1]指向第二个命令行参数，argv[2]指向第三个命令行参数……。

例子：

```
#include <stdio.h>
int main(int argc, char * argv[])
{
    for(int i = 0;i < argc; i ++ )
        printf( "%s\n",argv[i]);
    return 0;
}
```

将上面的程序编译成sample.exe，然后在控制台窗口敲:

`sample para1 para2 s.txt 5 4`

输出结果就是：

```
sample
para1
para2
s.txt
5
4
```

其他标准库函数

字符处理函数

在 `ctype.h` 中声明，主要有：

int isdigit(int c) 判断c是否是数字字符

int isalpha(int c) 判断c 是否是一个字母

int isalnum(int c) 判断c是否是一个数字或字母

int islower(int c) 判断 c 是否是一个小写字母

int islower(int c) 判断 c 是否是一个小写字母

int isupper(int c) 判断 c 是否是一个大写字母

int toupper(int c) 如果 c 是一个小写字母，则返回其大写字母

int tolower (int c) 如果 c 是一个大写字母，则返回其小写字母

字符串和内存操作函数

字符串和内存操作函数声明在 `string.h` 中，常用的有：

char * strchr(char * s, int c) 如果s中包含字符c,则返回一个指向s第一次出现的该字符的指针,否则返回NULL

char * strstr(char * s1, char * s2) 如果s2是s1的一个子串，则返回一个指向s1中首次出现s2的位置的指针，否则返回NULL

char * strlwr(char * s) 将s中的字母都变成小写

char * strupr(char * s) 将s中的字母都变成大写

char * strcpy(char * s1, char * s2) 将字符串s2的内容拷贝到s1中去

char * strncpy(char * s1, char * s2,int n) 将字符串s2的内容拷贝到s1中去，但是最多拷贝n个字节。如果拷贝字节数达到n，那么就不会往s1中写入结尾的'\0'

char * strcat(char * s1, char * s2) 将字符串s2添加到s1末尾

int strcmp(char * s1, char * s2) 比较两个字符串，大小写相关。如果返回值小于0，则说明s1按字典顺序在s2前面；返回值等于0，则说明两个字符串一样；返回值大于0，则说明s1按字典顺序在s2后面。

int stricmp(char * s1, char * s2) 比较两个字符串，大小写无关。其他和strcmp同。

void * memcpy(void * s1, void * s2, int n) 将内存地址s2处的n字节内容拷贝到内存地址s1

void * memset(void * s, int c, int n)　将内存地址s开始的n个字节全部置为c

字符串转换函数

有几个函数，可以完成将字符串转换为整数，或将整数转换成字符串等这类功能。它们定义在 `stdlib.h` 中:

int atoi(char *s)　将字符串s里的内容转换成一个整型数返回。比如，如果字符串s的内容是“1234”，那么函数返回值就是1234

double atof(char *s)　将字符串s中的内容转换成浮点数。

char *itoa(int value, char *string, int radix)　将整型值value以radix进制表示法写入 string。比如：

```
char szValue[20];
itoa(32,szValue,10); //则使得szValue的内容变为 “32”
itoa(32,szValue,16); //则使得szValue的内容变为 “20”
```

简单计算、数制转换、日期处理

例题： Package，校门外的数，进制确定

日历问题

在我们现在使用的日历中, 闰年被定义为能被4整除的年份，但是能被100整除，而不能被400整除的年是例外，它们不是闰年。例如：1700, 1800, 1900 和 2100 不是闰年，而 1600, 2000 和 2400是闰年。给定从公元2000年1月1日开始逝去得天数，你的任务是给出这一天是哪年哪月哪日星期几。

此题为典型的日期处理程序，没有难度,只是编程需要特别细心,日期处理的程序非常容易出错。

基本思路： 确定星期几；确定年；闰年366天，否则365天；确定月，每个月长短不同；确定日。

字符串处理

（1）strcpy在复制字符串str2到str1时，不检查str2是否超出了str1的存储容量，而是直接将str2中存储的字符串复制到从str1开始的一段连续区域，在程序中要特别注意这种情况所引发的程序运行不确定性。

（2）strlen()是一个o(N)的函数，每次判断 i < strlen(s) – 1 都要执行，太浪费时间了。

（3）stricmp()不区分大小写。

（4） `char *strstr(char *s1, char *s2);` 查找给定字符串在字符串中第一次出现的位置，返回位置指针（注意这是返回一个位置指针，而不是 `find()` 返回的位置数！用 `dest-begin+1` 确认位置）。

字符串部分拷贝 strncpy()

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
```

```
    char s1[20] = "1234567890";
    char s2[] =  "abcd" ;
    strncpy( s1,s2,5);
    cout << s1 << endl;
    strcpy( s1,"1234567890");
    strncpy( s1,s2,4);
    cout << s1 << endl;
    return ;
}
```

输出：

```
abcd
abcd567890
```

折半查找库函数

该函数与 qsort有类似之处

```
void *bsearch(const void *key, const void *base, size_t nelem, size_t width, int ( *fcmp)(const void *,
const void *));
```

该函数在一个**排过序的**数组base里查找值和* key 一样大(未必是严格相等，只是按 fcmp指定的比较规则是一样大的而已） 的元素。找到的话，返回第一个该种元素的地址，找不到的话，返回 NULL。

复杂度：O(lgN) 要求比较大小的规则和排序的规则一样

高精度计算

大整数加法

思路

1) 用字符型或整型数组来存放大整数

an[0]存放个位数，an[1]存放十位数，an[2]存放百位数……

2）模拟小学生列竖式做加法，从个位开始逐位相加，超过或达到10则进位。

用unsigned an1[201]保存第一个数，用unsigned an2[200]表示第二个数，然后逐位相加，相加的结果直接存放在an1中。要注意处理进位。

```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 201
int an1[MAX_LEN+10];
int an2[MAX_LEN+10];
char szLine1[MAX_LEN+10];
```

```
char szLine2[MAX_LEN+10];
int Add(int nMaxLen, int* an1, int* an2)
//将长度最多为 nMaxLen 的大整数 an1和an2 相加，结果放在an1，
//an1[0],an2[0]对应于个位
{
    int nHighestPos = 0;
    for(int i = 0;i < nMaxLen; i ++ ) {
        an1[i] += an2[i];       //逐位相加
        if( an1[i] >= 10 ) {        //看是否要进位
            an1[i] -= 10;
            an1[i+1] ++;            //进位
        }
        if( an1[i] )
            nHighestPos = i; //记录最高位的位置
    }
    return nHighestPos;
}
int main(void)   {
    scanf("%s", szLine1);  scanf("%s", szLine2);
    int i, j;
    //库函数memeset将地址an1开始的sizeof(an1)字节内容置成0
    //sizeof(an1)的值就是an1的长度
    memset( an1, 0, sizeof(an1));
    memset( an2, 0, sizeof(an2));
    //下面将szLine1中存储的字符串形式的整数转换到an1中去，
    //an1[0]对应于个位
    int nLen1 = strlen(szLine1);
    for(j =0, i = nLen1 - 1;i >= 0 ; i --)
        an1[j++] = szLine1[i] - '0';
    int nLen2 = strlen(szLine2);
    for(j=0, i = nLen2 - 1;i >= 0 ; i --)
        an2[j++] = szLine2[i] - '0';
    int nHighestPos = Add(MAX_LEN,an1,an2);
    for( i = nHighestPos; i >= 0; i -- )  printf("%d", an1[i]);
    //反过来一位一位输出
    return 0;
}
```

大整数减法

注意：默认$a > b$。

```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 110
int an1[MAX_LEN];
int an2[MAX_LEN];
char szLine1[MAX_LEN];
char szLine2[MAX_LEN];
int Substract( int nMaxLen, int * an1, int * an2)
{ //两个最多nMaxLen位的大整数an1减去an2， an1保证大于an2
    int nStartPos = 0;//注意是反过来的哈，向右边的借位
```

```
    for( int i = 0;i < nMaxLen ; i ++ ) {
        an1[i] -= an2[i];  //逐位减
        if( an1[i] < 0 ) { //看是否要借位
            an1[i] += 10;
            an1[i+1] --; //借位
        }
        if( an1[i] )
            nStartPos = i; //记录最高位的位置
    }
    return nStartPos; //返回值是结果里面最高位的位置
}
int main(void)
{
    int n;
    scanf("%d",&n);
    while(n -- ) {
        scanf("%s", szLine1);
        scanf("%s", szLine2);
        int i, j;
        memset( an1, 0, sizeof(an1));
        memset( an2, 0, sizeof(an2));
        //下面将szLine1中存储的字符串形式的整数转换到an1中去,
        //an1[0]对应于个位
        int nLen1 = strlen( szLine1);
        for(j = 0, i = nLen1 - 1;i >= 0 ; i --)
            an1[j++] = szLine1[i] - '0';
        int nLen2 = strlen(szLine2);
        for( j = 0, i = nLen2 - 1;i >= 0 ; i --)
            an2[j++] = szLine2[i] - '0';
        int nStartPos = Substract( MAX_LEN, an1,an2);
        for( i = nStartPos; i >= 0; i -- )
            printf("%d", an1[i]);
        printf("\n");
    }
    return 0;
}
```

• 大整数乘法

用 `unsigned an1[200]` 和 `unsigned an2[200]` 分别存放两个乘数，用 `aResult[400]` 来存放积。计算的中间结果也都存在 `aResult` 中。 `aResult` 长度取 `400` 是因为两个 `200` 位的数相乘，积最多会有 `400` 位。 `an1[0]`，`an2[0]`，`aResult[0]` 都表示个位。

一个数的第i位和另一个数的第j位相乘所得的数，一定是要累加到结果的第$i+j$位上。这里i,j都是从右往左，从0开始数。

计算的过程基本上和小学生列竖式做乘法相同。为编程方便，并不急于处理进位，而将进位问题留待最后统一处理。

```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 200
unsigned an1[MAX_LEN+10];
unsigned an2[MAX_LEN+10];
```

```
unsigned aResult[MAX_LEN * 2 + 10];
char szLine1[MAX_LEN+10];
char szLine2[MAX_LEN+10];
int main(void)
{
    gets( szLine1); //gets函数读取一行
    gets( szLine2);
    int i, j;
    int nLen1 = strlen( szLine1);
    memset( an1, 0, sizeof(an1));
    memset( an2, 0, sizeof(an2));
    memset( aResult, 0, sizeof(aResult));
    j = 0;
    for( i = nLen1 - 1;i >= 0 ; i --)
        an1[j++] = szLine1[i] - '0';
    int nLen2 = strlen(szLine2);
    j = 0;
    for( i = nLen2 - 1;i >= 0 ; i --)
        an2[j++] = szLine2[i] - '0';//j，i互补，相当于做一个反转
    //每一轮都用an1的一位，去和an2各位相乘，从an1的个位开始
    for( i = 0;i < nLen2; i ++ )    {
        //用选定的an1的那一位，去乘an2的各位
        for( j = 0; j < nLen1; j ++ )
            //两数第i，j位相乘，累加到结果的第i+j位
            aResult[i+j] += an2[i]*an1[j]; //important
    }
    //下面的循环统一处理进位问题
    int nHighestPos = 0;
    for( i = 0; i < MAX_LEN * 2; i ++ ) {
        if( aResult[i] >= 10 ) {
            aResult[i+1] += aResult[i] / 10;
            aResult[i] %= 10;
        }
        if( aResult[i] )    nHighestPos = i;
    }
    for( i = nHighestPos; i >= 0; i -- )
        printf("%d", aResult[i]);
    return 0;
}
```

大整数除法

基本的思想是反复做减法，看看从被除数里最多能减去多少个除数，商就是多少。一个一个减显然太慢，如何减得更快一些呢？以7546除以23为例来看一下：开始商为0。先减去23的100倍，就是2300，发现够减3次，余下646。于是商的值就增加300。然后用646减去230，发现够减2次，余下186，于是商的值增加20。最后用186减去23，够减8次，因此最终商就是328。

为什么是100？ `sizeof(7546)-sizeof(23)` 次幂。

所以本题的核心是要写一个大整数的减法函数，然后反复调用该函数进行减法操作。

> “
>
> Input
>
> 第1行是测试数据的组数n，每组测试数据占2行，第1行是被除数，第2行是除数。每组测试数据之间有一个空行，每行数据不超过100个字符
>
> Output
>
> n行，每组测试数据有一行输出是相应的整数商

```
#include <stdio.h>
#include <string.h>
#define MAX_LEN 110
int an1[MAX_LEN];
int an2[MAX_LEN];
int tmpAn2[MAX_LEN];
int anResult[MAX_LEN];
char szLine1[MAX_LEN];
char szLine2[MAX_LEN];
char szNouse[MAX_LEN];
int Substract( int nMaxLen, int * an1, int * an2)
   //大整数减法
}
int Length( int nMaxLen,int * an)
//求大整数的位数。0 算 0 位
{
    int i;
    for( i = nMaxLen -1 ; an[i] == 0 && i >= 0; i -- );
    if( i >= 0 )
        return i + 1;
    return 0;
}
void ShiftLeft( int nMaxLen,int * an1, int * an2, int n)
//将大整数an1左移n位，即乘以10的n次方，结果放到an2里
{
    memcpy( an2,an1,nMaxLen * sizeof(int));
    if( n <= 0 )
        return;
    for( int i = nMaxLen -1 ; i >= 0; i -- )
        if( i - n >= 0)
            an2[i] = an1[i-n];
        else
            an2[i] = 0;
}
int * Max(int nMaxLen, int * an1, int * an2)
//求大整数an1和an2里面大的那个
//如果都是0，则返回NULL
{
    //注意反序即可，没啥好说的
}
int main(void)
{
    int n;
    scanf("%d",&n);
```

```
    gets(szNouse);
    while(n -- ) {
        gets(szLine1);
        gets(szLine2);
        gets(szNouse);
        int i, j;
        //库函数memeset将地址an1开始的sizeof(an1)字节内容置成0
        //sizeof(an1)的值就是an1的长度
        //memset函数在string.h中声明
        memset( an1, 0, sizeof(an1));
        memset( an2, 0, sizeof(an2));
        //下面将szLine1中存储的字符串形式的整数转换到an1中去,
        //an1[0]对应于个位
        int nLen1 = strlen( szLine1);
        for(j = 0, i = nLen1 - 1;i >= 0 ; i --)
            an1[j++] = szLine1[i] - '0';
        int nLen2 = strlen(szLine2);
        for(j = 0, i = nLen2 - 1;i >= 0 ; i --)
            an2[j++] = szLine2[i] - '0';
        int nHighestPos = 0;
        memset(anResult,0,sizeof(anResult));
        int nShiftLen = Length(MAX_LEN,an1) - Length(MAX_LEN,an2);
        //只要an1大于an2，就不停相减
        while( Max(MAX_LEN,an1,an2) == an1 ) {
            //算出an1的10的nShiftLen次方倍
            ShiftLeft(MAX_LEN, an2, tmpAn2,nShiftLen);
            //重复减去an1的10的nShiftLen次方倍，看能减几次
            while( Max(MAX_LEN,an1,tmpAn2) == an1) {
                Substract(MAX_LEN, an1,tmpAn2);
                //记录商对应位
                anResult[nShiftLen] ++;
            }
            //记录结果最高位的位置
            if( nHighestPos == 0 && anResult[nShiftLen])
                nHighestPos = nShiftLen;
            nShiftLen --;//降级
        }
        for( i = nHighestPos; i >= 0; i -- )
            printf("%d", anResult[i]);
        printf("\n");
    }
    return 0;
}
```

(全局范围)

```cpp
#include<iostream>
#include<vector>
#include<ctime>

using namespace std;

vector<vector<string>> solution(int n);
int cnt = 0;
void backtrack(vector<string>& board, int row);
bool isValid(vector<string>& board, int row, int col);

vector<vector<string>> res;

/*输入棋盘边长，返回所有合法的放置方法*/

vector<vector<string>> solution(int n) {
    vector<string> borad(n, string(n, '_'));
    backtrack(borad, 0);
    return res;
}

void backtrack(vector<string>& board, int row) {
    if (row == board.size()) {
        res.push_back(board);
        return;
    }
    int n = board[row].size();
    for (int col = 0; col < n; col++) {
        if (!isValid(board, row, col)) {
            board[row][col] = 'Q';
            backtrack(board, n + 1);
            board[row][col] = '_';
        }
    }
    return;
}
```

```cpp
bool isValid(vector<string>& board, int row, int col) {
    int n = board[row].size();
    for (int i = 0; i <= row; i++) {
        if (board[i][col] == 'Q') {
            return false;
        }
    }
    for (int i = row - 1, int j = col - 1; i >= 0, j >= 0; i--, j--) {
        if (board[i][j] == 'Q') {
            return false;
        }
    }
    for (int i = row - 1, int j = col + 1; i >= 0, j <= n; i--, j++) {
        if (board[i][j] == 'Q') {
            return false;
        }
    }
    return true;
}
```

C++考 非常简单

四道大题（程序设计二的考试内容*3：字符串处理，高精度，2的次方，模拟(没讲？)，枚举，搜索，递归，回溯；动态规划不考，60分）

C++，类，访问控制，友元，带static的函数不考；继承，继承的语法，派生（从类、类模板），公有继承，常量成员函数(不特别考)，构造函数（重载），析构函数，拷贝构造函数（形式），动态内存分配您不敢考，继承重写函数（静态绑定），模板（链表的例子）【注意】，函数重载，运算符重载，

选择（10*2）

8.2.5 拷贝构造函数

因为拷贝生成对象的方式为:

Howmany h2(h); h2=h;

所以拷贝构造函数的形式应该为: X(X&);

例如:

```
Howmany::Howmany(const Howmany &h)
{
  …
}
```

注意:

1. 参数必须是类howmany的const引用(const howmany &),why?(否则会引起无穷递归!)
2. 没有返回值。

公有继承

- 基类成员对派生类的可见性:
  --公有成员和保护成员可见，私有成员不可见。
- 基类成员对派生类对象的可见性
  --公有成员可见，保护成员和私有成员不可见。

12.3.1 类模板定义

```
template <class T>
class 类模板名
{
      // 类模板的定义
};
```

注：T是一个类模板的类型参数，可以有一个或多个，可以是任意类型。

12.3.2 类模板实例化

- 给类模板的参数指定具体的类型，这一过程称为“类模板的实例化”。

  类模板名<具体类型表>

```
Stack<int>;      // 实例化成 int 型栈类;
Stack<char>;     // 实例化成 char 型栈类;
```

注意：类模板实例化后的结果是类，而不是对象！