编程40题.md

1. 阅读教材P111至P113的例5.10关于斐波那契数列的讲解，理解斐波那契数列的两种递归实现方式，并实现P113中递归函数的改进版。

2. 阅读教材P113的例5.11关于汉诺塔的讲解，并实现此程序。

3. 台阶问题。

问题描述：
有N级台阶，刚开始时你在第一级，若每次只能跨上一级或二级，要走上第N级台阶，共有多少种走法？

输入：
输入数据为一个正整数N(1≤N≤1000)，表示台阶的级数。

输出：
输出走上第N级台阶的走法数目，由于答案可能很大，你需要输出走法数目对100007取余后的结果。

样例输入#1：
2
样例输出#1：
1

样例输入#2：
3
样例输出#2：
2

样例输入#3：
4
样例输出#3：
3

4. 台阶问题加强版。

问题描述：
有N级台阶，刚开始时你在第一级，若每次可以向上跨最多K级台阶，要走上第N级台阶，共有多少种走法？

输入：
输入数据为一个正整数N(1≤N≤1000)，表示台阶的级数；一个正整数K(1≤K≤100)，表示每次最多可以向上跨的台阶数。

输出：
输出走上第N级台阶的走法数目，由于答案可能很大，你需要输出走法数目对100007取余后的结果。

样例输入#1：
6 2

样例输出#1:
8

5. 汉诺塔加强版:汉诺双塔问题。

问题描述:
给定A、B、C三根足够长的细柱,在A柱上放有2n个中间有孔的圆盘,共有n个不同的尺寸,每个尺寸都有两个相同的圆盘,注意这两个圆盘是不加区分的(2018级软工专业群的图片是n为3时的情形)。
现要将这些圆盘移到C柱上,在移动过程中可放在B柱上暂存。要求:
(1)每次只能移动一个圆盘;
(2)A、B、C三根细柱上的圆盘都要保持上小下大的顺序。
求:设A(n)为2n个圆盘完成上述任务所需的最少移动次数,对于输入的n,输出A(n)。

输入:
输入数据为一个正整数n(1≤n≤25),表示在A柱上放有2n个圆盘。

输出:
输出完成上述任务所需的最少移动次数A(n)。

样例输入#1:
1
样例输出#1:
2

样例输入#2:
2
样例输出#2:
6

(提示:设法建立A(n)与A(n-1)的递推关系式)。

6. 汉诺塔加强加强版:有限制的汉诺塔问题。

经过以上两道汉诺塔问题,你已经很熟悉汉诺塔游戏的规则了。现在我们来对原始的汉诺塔加上一个限制:不能从最左侧的塔直接移动到最右侧,也不能从最右侧直接移动到最左侧,而是必须经过中间的塔。也就是说,如果你想将一个原盘从A塔,移动到C塔,你应当先将该原盘从A塔移动到B塔,再将该原盘从B塔移动到C塔。

基于以上限制,输出将n个原盘从A塔全部移动到C塔的过程,并在最后输出总共的移动步骤。n为读入的值。

7. 利用组合数的性质,使用递归方法求二项式系数C(n,k)。其中,组合数的性质为: C(n,k)=C(n-1,k-1)+C(n-1,k)。递归函数声明如下:

```
int BinomialCoefficient(int n, int k);
```

在主函数中调用该函数,并进行多组测试,以验证你的函数。

8. 写一个函数,函数声明如下:

```
int is_sorted(int arr[], int left, int right);
```

该函数接受三个参数，判断数组arr的子区间arr[left],arr[left+1],...,arr[right-1],arr[right]是否非严格单调递增,即对于任意的下标i，只要满足left<=i<=right-1，都有arr[i]<=arr[i+1]。在主函数中调用该函数，并进行多组测试，以验证你的函数。

9. 写一个函数，函数声明如下：

```
void BubbleSort(int arr[], int left, int right);
```

该函数接受三个参数，对数组arr的子区间arr[left],arr[left+1],...,arr[right-1],arr[right]，进行冒泡排序（升序排序）。在主函数中调用该函数，并进行多组测试，以验证你的函数（注：你可以采用生成随机数的方式随机生成若干数据进行测试）。

10. 实现二分查找，请分别给出循环实现与递归实现。函数声明如下：

```
int binary_search_recursive(int arr[], int left, int right, int query);
int binary_search_iterative(int arr[], int left, int right, int query);
```

该函数接受四个参数，binary_search_recursive为递归版本；binary_search_ iterative为循环（迭代）版本。数组arr的子区间arr[left],arr[left+1],...,arr[right-1],arr[right]严格单调递增,即对于任意的下标i，只要满足left<=i<=right-1，都有arr[i]<arr[i+1]，query为待查找的关键字。要求使用二分查找的方式，在该子区间内查找值为query的元素，并返回该元素的数组下标；若未找到，则返回-1。

11. 实现插入排序。函数声明如下：

```
void InsertionSort(int arr[], int left, int right);
```

该函数接受三个参数，实现插入排序的功能（升序排序）。参数的意义及要求同第9题。

12. 实现选择排序。函数声明如下：

```
void SelectionSort(int arr[], int left, int right);
```

该函数接受三个参数，实现选择排序的功能（升序排序）。参数的意义及要求同第9题。

13. 写一个函数，函数声明如下：

```
void k_reverse(char* str, int k);
```

该函数将字符串str中的每k个字符进行反转。举例来说，若str为"HelloWorld."，k为3，则反转后的结果为"leHWollrod."，注意，最后不足k个的字符不用反转。在主函数中调用该函数，并进行多组测试，以验证你的函数。

14. 实现高精度加法。

输入：输入数据为两个正整数A和B。A、B最大均为一千位。

输出：输出A+B的值。

（注：请不要使用浮点数进行运算。你应当使用数组表示A与B）。

15. 打印出螺旋矩阵。螺旋矩阵是一个N * N的方阵，其中元素为自然数，像螺旋方向一样递增。

```
N为3时的螺旋矩阵为：
1 2 3
8 9 4
7 6 5
N为4时的螺旋矩阵为：
1 2 3 4
12 13 14 5
11 16 15 6
10 9 8 7
N为5时的螺旋矩阵为：
1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9
以此类推。
```

输入：正整数N，N≤20。

输出：输出对应N阶的螺旋矩阵。

16. 实现教材P167任务7.3，可参考P174的7.3.5。

17. 实现自己版本的strlen。函数声明如下：

```
int my_strlen(const char* str);
```

类似于库文件中的strlen，该函数返回字符串str的长度。在主函数中调用该函数，并进行多组测试，以验证你的函数。

18. 实现自己版本的strcpy。函数声明如下：

```
void my_strcpy(char* dest, const char* src);
```

类似于库文件中的strcpy，该函数将字符串src复制给字符串dest，或者说，把从src地址开始且含有'\0'结束符的字符串复制到以dest开始的地址空间。在主函数中调用该函数，并进行多组测试，以验证你的函数。

19. 实现自己版本的strcmp。函数声明如下：

```
int my_strcmp(const char* str1, const char* str2);
```

类似于库文件中的strcmp，该函数比较字符串str1与str2的大小。若str1与str2完全相同，则返回0；若str1小于str2，则返回-1；若str1大于str2，则返回1。即，两个字符串自左向右逐个字符相比（按ASCII值大小相比较），直到出现不同的字符或遇'\0'为止。在主函数中调用该函数，并进行多组测试，以验证你的函数。

20. 写一个程序，读入n名学生的姓名、学号、成绩，分别输出成绩最高和成绩最低学生的姓名和学号。

输入：输入格式为：
第1行：正整数n
第2行：第1个学生的姓名 学号 成绩
第3行：第2个学生的姓名 学号 成绩
... ...
第n+1行：第n个学生的姓名 学号 成绩
其中姓名和学号均为不超过10个字符的字符串，成绩为0到100之间的一个整数，输入数据保证没有两个学生的成绩是相同的。

输出：对每个测试用例输出2行，第1行是成绩最高学生的姓名和学号，第2行是成绩最低学生的姓名和学号。

输入样例：
```
3
Joe Math990112 89
Mike CS991301 100
Mary EE990830 95
```
输出样例：
```
Mike CS991301
Joe Math990112
```

要求：请使用结构体完成此题。

21. 写一个程序，实现如下功能：可读取一个文本文件，并将该文件中的内容原封不动的在控制台打印出来（类似于Linux的cat命令的其中一个功能）。（涉及到文件读取的知识）。要求：你应当使用到main函数的形参。也就是说，若你的程序成功编译后的可执行文件为show.out，当前目录下有一个文本文件名为test.txt，则使用命令：

```
./show.out test.txt
```

即可打印出test.txt文件中的内容。

22. 写一个程序，实现如下功能：将一个文件拷贝到某一目录中（类似于Linux的cp指令的其中一个功能）。（涉及到文件读与写的知识）要求：你应当使用到main函数的形参。也就是说，若你的程序成功编译后的可执行文件为copy.out，则使用命令：

```
./copy.out <src_file> <dest_dir>
```

即可将文件src_file拷贝到dest_dir目录下。例如，如果想将文件 `/root/tcpl/example.c` 拷贝到 `/root` 目录下，则可使用命令：

```
./copy.out /root/tcpl/example.c /root
```

23. 奇约数问题。

题目描述：定义函数f(x)为x最大的奇数约数，其中，x为正整数。例如：f(44)=11。现给出整数N，请求出f(1)+f(2)+…+f(N)。例如，若N为7，则

f(1)+f(2)+f(3)+f(4)+f(5)+f(6)+f(7)=1+1+3+1+5+3+7=21。

输入:输入一个正整数N(1≤N≤1000000000)。

输出:输出一个整数,即f(1)+f(2)+…+f(N)。

样例输入:
7
样例输出:
21

24. 写一个函数,函数声明如下:

```
int bin_insert(int n, int m, int j, int i);
```

接受两个32位整数n和m,实现将m的二进制数位插入到n的二进制的第j到第i位,其中二进制的位数从低位数到高位且以0开始。返回操作后的数,保证n的第j到第i位均为零,且m的二进制位数小于等于i-j+1。举例来说,当n、m、j、i分别为1024、19、2、6时,返回1100。在主函数中调用该函数,并进行多组测试,以验证你的函数。

25. 写一个函数,函数声明如下:

```
int sum(int n);
```

接受正整数n,求1+2+3+…+n。要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句(A?B:C)。在主函数中调用该函数,并进行多组测试,以验证你的函数。

26. 写一个函数,函数声明如下:

```
int add(int n1, int n2);
```

接受两个整数n1和n2(未必为正整数),求n1和n2之和。要求不得使用 +、-、*、/ 四则运算符号。在主函数中调用该函数,并进行多组测试,以验证你的函数。

27. 写一个函数,函数声明如下:

```
int substr(const char* str, const char* substr);
```

接受两个字符串str和substr,其中,str为主串,substr为子串。返回substr在str中第一次出现的位置(数组下标);若substr尚未在str中出现,则返回-1。例如,若str为"I dislike C",substr为"like",则返回5;若str为"I dislike C",substr为"love",则返回-1。在主函数中调用该函数,并进行多组测试,以验证你的函数。

28. 写一个函数,函数声明如下:

```
int count_substr(const char* str, const char* substr);
```

接受两个字符串str和substr，其中，str为主串，substr为子串。返回substr在str中出现的次数。注意，"hh"在"hhh"中仅出现一次，而非两次；"hh"在"hhhh"中出现两次；"h"在"hhh"中出现三次；"sos"在"sososos"中仅出现两次，而非三次。在主函数中调用该函数，并进行多组测试，以验证你的函数。

29. 实现归并排序。函数声明如下：

```
void MergeSort(int arr[], int left, int right);
```

该函数接受三个参数，实现选择排序的功能（升序排序）。参数的意义及要求同第9题。

30. 学会使用库函数qsort。qsort是定义在头文件stdlib.h中的一个排序函数，它使用快速排序的方式。qsort的命名来自“quick sort”，即快速排序。你现在无须知道快速排序的实现细节，你只需学会使用这个函数即可。你可以参阅百度百科 或cppreference 中关于qsort的讲解。学会使用qsort，并实现以上cppreference链接中的例子。

31. 学会使用库函数bsearch。bsearch是定义在头文件stdlib.h中的一个查找函数，它使用二分查找的方式。bsearch的命名来自“binary search”，即二分查找。你已经清楚了二分查找的实现，如果你感兴趣，可以阅读bsearch的实现源码，以加深理解。你可以参阅百度百科 或cppreference 中关于bsearch的讲解。学会使用bsearch，并实现以上cppreference链接中的例子。

32. 链表结点类型定义如下：

```
struct Node {
        int val;
        struct Node* next;
};
```

写一个函数，函数声明如下：

```
struct Node* construct(int arr[], int size);
```

接受整型数组arr与其大小size。该函数根据数组arr，构造一个链表，并返回链表的头结点。注意，保持链表中元素的顺序与数组中元素的顺序一致。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。

33. 链表结点类型定义同第32题。写一个函数，函数声明如下：

```
struct Node* insert(struct Node* head, int val);
```

接受一链表的头结点head，以及一个元素值val。该函数以val为元素值创建一个新的节点，将该节点插入该链表的头部，并返回插入后的链表头节点。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。

34. 链表结点类型定义同第32题。写一个函数，函数声明如下：

```
struct Node* insert(struct Node* head, int val);
```

接受一链表的头结点head，以及一个元素值val。该函数以val为元素值创建一个新的节点，将该节点插入该链表的尾部，并返回插入后的链表头节点。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。

35. 链表结点类型定义同第32题。写一个函数，函数声明如下：

```
struct Node* insert(struct Node* head, int val);
```

接受一有序（非降序）链表的头结点head。该函数以val为元素创建一个新的节点，并将该节点插入到该链表合适的位置，以维持链表的有序性。函数返回插入后的链表头节点。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。

36. 链表结点类型定义同第32题。写一个函数，函数声明如下：

```
struct Node* delete(struct Node* head, struct Node* target);
```

接受一链表的头结点head，以及该链表中的某节点target。该函数将target节点从该链表中删除，并返回删除后的链表头结点；若删除后链表为空，返回NULL。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数（注：target节点有可能就是头结点，须注意）。

37. 链表结点类型定义同第32题。写一个函数，函数声明如下：

```
struct Node* delete(struct Node* head, int val);
```

接受一链表的头结点head，以及一个元素值val。该函数将该链表中所有元素值为val的节点删除，并返回删除后的链表头结点；若删除后链表为空，返回NULL。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。

38. 链表结点类型定义同第32题。写一个函数，函数声明如下：

```
struct Node* reverse(struct Node* head);
```

接受一链表的头结点head。该函数将该链表进行原地（in-place）反转（即不允许重新创建一个新的链表），并返回反转后链表的头节点。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。

39. 在归并排序的实现中，你已经了解了将两个有序数组有序归并为一个更大数组的方法。现在，我们来对两个有序链表进行有序归并。链表结点类型定义同第1题。写一个函数，函数声明如下：

```
struct Node* merge(struct Node* head1, struct Node* head2);
```

分别接受两个有序（非降序）链表的头结点，分别为head1与head2。该函数实现将这两个链表进行有序（非降序）归并到一个新的链表中，并返回所归并后的新链表的头节点。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。

40. 链表结点类型定义同第32题。写一个函数，函数声明如下：

```
void clear(struct Node* head);
```

接受一链表的头结点head。该函数将该链表清空，即删除所有节点，并释放其内存空间。写出完整的程序，在主函数中调用该函数，并进行多组测试，以验证你的函数。