2015 年决赛 C/C++大学 A 组

考生须知：

- 考试开始后，选手首先下载题目，并使用考场现场公布的解压密码解压试题。
- 考试时间为 4 小时。时间截止后，提交答案无效。
- 在考试强制结束前，选手可以主动结束考试（需要身份验证），结束考试后将无法继续提交或浏览答案。
- 选手可浏览自己已经提交的答案。被浏览的答案允许拷贝。
- 对同一题目，选手可多次提交答案，以最后一次提交的答案为准。
- 选手切勿在提交的代码中书写“姓名”、“考号”，“院校名”等与身份有关的信息或其它与竞赛题目无关的内容，否则成绩无效。
- 选手必须通过浏览器方式提交自己的答案。选手在其它位置的作答或其它方式提交的答案无效。
- 试题包含三种类型：“结果填空”、“代码填空”与“程序设计”。

结果填空题：

要求选手根据题目描述直接填写结果。**求解方式不限**。不要求源代码。

答案直接通过网页提交即可。不要书写多余的内容。

代码填空题：

要求选手在弄清给定代码工作原理的基础上，填写缺失的代码，使得程序逻辑正确。

所填写的代码不超过一条语句（即中间不能出现分号）。

把答案（仅填空处的答案，**不包括题面已存在的代码或符号**）直接通过网页提交。

不要书写多余的内容（比如注释）。

使用 ANSI C/ANSI C++ 标准，不要依赖操作系统或编译器提供的特殊函数。

程序设计题目：

要求选手设计的程序对于给定的输入能给出正确的输出结果。

考生的程序只有能运行出正确结果才有机会得分。

注意：在评卷时使用的输入数据与试卷中给出的示例数据可能是不同的。**选手的程序必须是通用的**，不能只对试卷中给定的数据有效。

要求选手给出的解答完全符合 ANSI C++标准，不能使用诸如绘图、Win32API、中断调用、硬件操作或与操作系统相关的 API。

代码中允许使用 STL 类库，但不能使用 MFC 或 ATL 等非 ANSI C++标准的类库。例如，不能使用 CString 类型（属于 MFC 类库）。

注意：main 函数必须返回 0

注意：所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。

所有源码必须在同一文件中。调试通过后，拷贝提交。

提交时，注意选择所期望的编译器类型。

1. 结果填空 (满分 19 分)

问题的描述在考生文件夹下对应题号的“题目.txt”中。相关的参考文件在同一目录中。

不限解决问题的方式或工具，只要求结果。

只能通过浏览器提交答案。

2. 结果填空 (满分 35 分)

问题的描述在考生文件夹下对应题号的“题目.txt”中。相关的参考文件在同一目录中。
要求参见前一题。

3. 代码填空 (满分 31 分)

问题的描述在考生文件夹下对应题号的“题目.txt”中。相关的参考文件在同一目录中。
填写的代码必须符合 ANSI C/C++ 标准。
代码不能只对题面特殊数据有效，应当具有通用性。
不要填写多余内容（如：题面上已存在的代码或符号）。
只能通过浏览器提交答案。

4. 程序设计（满分 41 分）

问题的描述在考生文件夹下对应题号的“题目.txt”中。相关的参考文件在同一目录中。
在评卷时使用的输入数据与试卷中给出的示例数据可能是不同的。**选手的程序必须是通用的**，不能只对试卷中给定的数据有效。
仔细阅读程序的输入、输出要求，千万不要输出没有要求的、多余的内容，例如：“请您输入 xx 数据：”。
建议仔细阅读示例，不要想当然!
处理完一个用例的数据后，立即退出（return 0），不要循环等待下一个用例的输入。
程序必须使用标准输入、标准输出，以便于机器评卷时重定向。
要求选手给出的解答完全符合 ANSI C/C++标准，不能使用诸如绘图、Win32API、中断调用、硬件操作或与操作系统相关的 API。
代码中允许使用 STL 类库，但不能使用 MFC 或 ATL 等非 ANSI C++标准的类库。例如，不能使用 CString 类型（属于 MFC 类库）。
注意: main 函数结尾需要 return 0
注意：所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。
所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。
提交时，注意选择所期望的编译器类型。

5. 程序设计（满分 75 分）

问题的描述在考生文件夹下对应题号的“题目.txt”中。相关的参考文件在同一目录中。
要求参见前一题。

6. 程序设计（满分 99 分）

问题的描述在考生文件夹下对应题号的“题目.txt”中。相关的参考文件在同一目录中。
要求参见前一题。

标题：方格填数

在 2 行 5 列的格子中填入 1 到 10 的数字．
要求：
相邻的格子中的数，右边的大于左边的，下边的大于上边的．

如【图 1.png】所示的 2 种，就是合格的填法．

请你计算一共有多少种可能的方案．

请提交该整数，不要填写任何多余的内容（例如：说明性文字）．

1	2	3	5	8
4	6	7	9	10

1	3	5	6	7
2	4	8	9	10

标题：四阶幻方

把 1~16 的数字填入 4x4 的方格中，使得行、列以及两个对角线的和都相等，满足这样的特征时称为：四阶幻方．

四阶幻方可能有很多方案．如果固定左上角为 1，请计算一共有多少种方案．
比如：

```
 1  2 15 16
12 14  3  5
13  7 10  4
 8 11  6  9
```

以及：

```
 1 12 13  8
 2 14  7 11
15  3 10  6
16  5  4  9
```

就可以算为两种不同的方案.

请提交左上角固定为 1 时的所有方案数字，不要填写任何多余内容或说明文字.

标题：显示二叉树

排序二叉树的特征是:
某个节点的左子树的所有节点值都不大于本节点值.
某个节点的右子树的所有节点值都不小于本节点值.

为了能形象地观察二叉树的建立过程，小明写了一段程序来显示出二叉树的结构来.

```
#include <stdio.h>
#include <string.h>
#define N 1000
#define HEIGHT 100
#define WIDTH 1000

struct BiTree
{
    int v;
    struct BiTree* l;
    struct BiTree* r;
};

int max(int a, int b)
{
    return a>b? a : b;
}

struct BiTree* init(struct BiTree* p, int v)
{
    p->l = NULL;
    p->r = NULL;
    p->v = v;

    return p;
}

void add(struct BiTree* me, struct BiTree* the)
{
```

```
    if(the->v < me->v){
        if(me->l==NULL) me->l = the;
        else add(me->l, the);
    }
    else{
        if(me->r==NULL) me->r = the;
        else add(me->r, the);
    }
}

//获得子树的显示高度
int getHeight(struct BiTree* me)
{
    int h = 2;
    int hl = me->l==NULL? 0 : getHeight(me->l);
    int hr = me->r==NULL? 0 : getHeight(me->r);

    return h + max(hl,hr);
}

//获得子树的显示宽度
int getWidth(struct BiTree* me)
{
    char buf[100];
    sprintf(buf,"%d",me->v);
    int w = strlen(buf);
    if(me->l) w += getWidth(me->l);
    if(me->r) w += getWidth(me->r);
    return w;
}

int getRootPos(struct BiTree* me, int x){
    return me->l==NULL? x : x + getWidth(me->l);
}

//把缓冲区当二维画布用
void printInBuf(struct BiTree* me, char buf[][WIDTH], int x, int y)
{
    int p1,p2,p3,i;
    char sv[100];
    sprintf(sv, "%d", me->v);

    p1 = me->l==NULL? x : getRootPos(me->l, x);
    p2 = getRootPos(me, x);
```

```
    p3 = me->r==NULL? p2 : getRootPos(me->r, p2+strlen(sv));

    buf[y][p2] = '|';
    for(i=p1; i<=p3; i++) buf[y+1][i]='-';
    for(i=0; i<strlen(sv); i++) buf[y+1][p2+i]=sv[i];
    if(p1<p2) buf[y+1][p1] = '/';
    if(p3>p2) buf[y+1][p3] = '\\';

    if(me->l) printInBuf(me->l,buf,x,y+2);
    if(me->r) _____________________________________;   //填空位置
}

void showBuf(char x[][WIDTH])
{
    int i,j;
    for(i=0; i<HEIGHT; i++){
        for(j=WIDTH-1; j>=0; j--){
            if(x[i][j]==' ') x[i][j] = '\0';
            else break;
        }
        if(x[i][0])printf("%s\n",x[i]);
        else break;
    }
}

void show(struct BiTree* me)
{
    char buf[HEIGHT][WIDTH];
    int i,j;
    for(i=0; i<HEIGHT; i++)
    for(j=0; j<WIDTH; j++) buf[i][j] = ' ';

    printInBuf(me, buf, 0, 0);
    showBuf(buf);
}

int main()
{
    struct BiTree buf[N];     //存储节点数据
    int n = 0;                //节点个数
    init(&buf[0], 500); n++;   //初始化第一个节点

    add(&buf[0], init(&buf[n++],200));   //新的节点加入树中
    add(&buf[0], init(&buf[n++],509));
```

```
    add(&buf[0], init(&buf[n++],100));
    add(&buf[0], init(&buf[n++],250));
    add(&buf[0], init(&buf[n++],507));
    add(&buf[0], init(&buf[n++],600));
    add(&buf[0], init(&buf[n++],650));
    add(&buf[0], init(&buf[n++],450));
    add(&buf[0], init(&buf[n++],440));
    add(&buf[0], init(&buf[n++],220));

    show(&buf[0]);
    return 0;
}
```

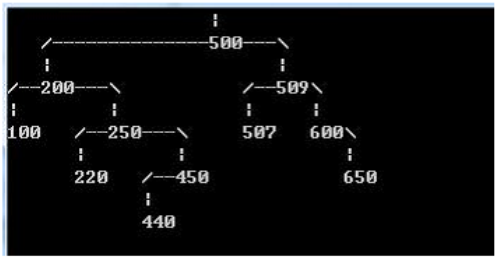
对于上边的测试数据，应该显示出：

```
                    |
   /--------------500---\
   |                      |
/--200---\              /--509\
|        |              |     |
100   /--250---\       507   600\
      |        |                |
      220   /--450             650
            |
            440
```

（如有对齐问题，请参考【图 1.png】）

请分析程序逻辑，填写划线部分缺失的代码。

注意，只填写缺少的部分，不要填写已有的代码或符号，也不要加任何说明文字。



标题：穿越雷区

X 星的坦克战车很奇怪，它必须交替地穿越正能量辐射区和负能量辐射区才能保持正常运转，否则将报废。

某坦克需要从 A 区到 B 区去（A，B 区本身是安全区，没有正能量或负能量特征），怎样走才能路径最短？

已知的地图是一个方阵，上面用字母标出了 A，B 区，其它区都标了正号或负号分别表示正负能量辐射区。

例如：

A + - + -
- + - - +
- + + + -
+ - + - +
B + - + -

坦克车只能水平或垂直方向上移动到相邻的区。

数据格式要求：

输入第一行是一个整数 n，表示方阵的大小， 4<=n<100
接下来是 n 行，每行有 n 个数据，可能是 A，B，+，-中的某一个，中间用空格分开。
A，B 都只出现一次。

要求输出一个整数，表示坦克从 A 区到 B 区的最少移动步数。
如果没有方案，则输出-1

例如：
用户输入：
5
A + - + -
- + - - +
- + + + -
+ - + - +
B + - + -

则程序应该输出：
10

资源约定：
峰值内存消耗 < 512M
CPU 消耗 < 1000ms

请严格按要求输出，不要画蛇添足地打印类似：“请您输入...” 的多余内容。

所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。

注意: main 函数需要返回 0
注意：只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意：所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。

提交时，注意选择所期望的编译器类型。

标题：切开字符串

Pear 有一个字符串，不过他希望把它切成两段。
这是一个长度为 N（<=10^5）的字符串。
Pear 希望选择一个位置，把字符串不重复不遗漏地切成两段，长度分别是 t 和 N-t（这两段都必须非空）。

Pear 用如下方式评估切割的方案：
定义“正回文子串”为：长度为奇数的回文子串。
设切成的两段字符串中，前一段中有 A 个不相同的正回文子串，后一段中有 B 个不相同的非正回文子串，则该方案的得分为 A*B。

注意，后一段中的 B 表示的是：“...非正回文...”，而不是：“...正回文...”。
那么所有的切割方案中，A*B 的最大值是多少呢?

【输入数据】
输入第一行一个正整数 N（<=10^5）
接下来一行一个字符串，长度为 N。该字符串仅包含小写英文字母。
【输出数据】
一行一个正整数，表示所求的 A*B 的最大值。
【样例输入】
10
bbaaabcaba
【样例输出】
38
【数据范围】

对于 20%的数据，N<=100
对于 40%的数据，N<=1000
对于 100%的数据，N<=10^5

资源约定：
峰值内存消耗 < 512M
CPU 消耗 < 1000ms

请严格按要求输出，不要画蛇添足地打印类似：“请您输入...” 的多余内容。

所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。

注意: main 函数需要返回 0
注意：只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意：所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。

提交时，注意选择所期望的编译器类型。

标题：铺瓷砖

为了让蓝桥杯竞赛更顺利的进行，主办方决定给竞赛的机房重新铺放瓷砖。机房可以看成一个 n*m 的矩形，而这次使用的瓷砖比较特别，有两种形状，如【图 1.png】所示。在铺放瓷砖时，可以旋转。

主办方想知道，如果使用这两种瓷砖把机房铺满，有多少种方案。

【输入格式】
输入的第一行包含两个整数，分别表示机房两个方向的长度。

【输出格式】
输出一个整数，表示可行的方案数。这个数可能很大，请输出这个数除以 65521 的余数。

【样例输入 1】
4 4
【样例输出 1】
2
【样例说明 1】
这两种方案如下【图 2.png】所示：

【样例输入 2】
2 6
【样例输出 2】
4
【数据规模与约定】
对于 20%的数据，1<=n, m<=5。
对于 50%的数据，1<=n<=100，1<=m<=5。
对于 100%的数据，1<=n<=10^15，1<=m<=6。

资源约定：
峰值内存消耗 < 512M
CPU 消耗 < 5000ms

请严格按要求输出，不要画蛇添足地打印类似：“请您输入...” 的多余内容。

所有代码放在同一个源文件中，调试通过后，拷贝提交该源码。

注意: main 函数需要返回 0
注意：只使用 ANSI C/ANSI C++ 标准，不要调用依赖于编译环境或操作系统的特殊函数。
注意：所有依赖的函数必须明确地在源文件中 #include <xxx>， 不能通过工程设置而省略常用头文件。

提交时，注意选择所期望的编译器类型。