## Brute force counting – 3n + 1 problem    (C)

```c
/* @JUDGE_ID: 14703TH 100 C "brute force" */

/* Tim Singer - Fall 2001 */

/* 3n+1 problem */

#include <stdio.h>

/* findcyc: run the program and count the cycle length as you go */
int findcyc(unsigned long n) {   /* input n */
  int cyc;
  cyc=0;
  do {
    cyc++;
    if(n==1) return cyc;         /* if n = 1 then stop */
    if(n&1) {                    /* if n = odd then n <- 3n + 1 */
      n=3*n+1;
    } else {                     /* else n <- n/2 */
      n/=2;
    }
  } while(1); /* forever */      /* goto 2 */
}

int main() {
  unsigned long i,j,nexti,n,curcyc,maxcyc,k;

  do {
    maxcyc=0;
   /* read i and j */
    scanf("%li",&i);
    scanf("%li",&j);

    if(feof(stdin)) break;  /* detect past end of file */

   /* print i and j */
    printf ("%li %li ",i,j);

   /* swap i and j if j is less */
    if(j<i) {
      k=i; i=j; j=k;
    }

   /* find the cycles for N=i..j, inclusive */
    for(n=i;n<=j;n++) {
     curcyc=findcyc(n);
     if(curcyc>maxcyc) maxcyc=curcyc;
    }

    printf ("%li\n",maxcyc);

  } while(!feof(stdin));
  return 1;
}
```

## Calculate nth root of p directly (C)

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* @JUDGE_ID: 14709PJ 113 C */
/* Dan Stutzbach - Fall 2001 */

/* The given problem is so solve the equation k^n = p, for k, given n and p */

/* Algebra:

   k^n = p
   ln (k^n) = ln (p)
   n * ln (k) = ln (p)
   ln (k) = ln (p) / n

   k = e^(ln (p) / n)

   Therefore, we just compute k directly and print out the answer.
*/


int main (void) {
        char line[256];
        double n;
        double p;
        double k;

        /* We are given n on a line, followed by p on a line.  We are
           to print out the solution, then read another pair until EOF */

        /* while (fgets (...)) is a handy way to read until end of
           file; at the end, fgets will return NULL.  Normally it
           returns a pointer to it's first argument ("line" in this
           case) */

        while (fgets (line, sizeof line, stdin)) {
                n = strtoul (line, NULL, 0);
                fgets (line, sizeof line, stdin);
                p = strtod (line, NULL);
                k = exp (log (p) / n);

                /* Note: casting k to an int will give the wrong
                   answer if k is 0.999.  Casting to int _ALWAYS_
                   rounds down!  Instead, we use printf, and tell it
                   to round off to 0 decimal places.  This is a much
                   safer way to convert from a double/float to an
                   int.
                */
                printf ("%0.0f\n", k);
        }

        return 0;
}
```

## Clock patience – card game (C)

```
#include <stdio.h>

/* @JUDGE_ID: 14709PJ 170 C */
/* Dan Stutzbach - Fall 2001 */

/* This program plays a silly little card game */

/* We store each card as an int.  The low-order bits are the value of
   card, and the high-order bits are the suit.  Ace is low. */

/* Masks to extract the suit or value */
#define SUIT_MASK 0xf000
#define VALUE_MASK 0x0fff
enum suit {
        HEART = 0x1000,
        DIAMOND = 0x2000,
        CLUB = 0x3000,
        SPADE = 0x4000
};

/* Convert an integer card into a printable suit */
char to_suit (int card)
{
        switch ((enum suit) (card & SUIT_MASK)) {
        case HEART:   return 'H';
        case DIAMOND: return 'D';
        case CLUB:    return 'C';
        case SPADE:   return 'S';
        }
        return -1;
}

/* Convert an integer card into a printable value */
char to_value (int card)
{
        card &= VALUE_MASK;
        switch (card) {
        case 1:   return 'A';
        case 2: case 3: case 4: case 5: case 6: case 7: case 8: case 9:
                return card + '0';
        case 10:  return 'T';
        case 11:  return 'J';
        case 12:  return 'Q';
        case 13:  return 'K';
        }
        return -1;
}

/* Convert a printable value and suit into an integer card */
int to_card (char value, char suit)
{
        int card;
        switch (value) {
        case 'A': card = 1; break;
        case '2': case '3': case '4': case '5': case '6': case '7':
        case '8': case '9':
                card = value - '0'; break;
        case 'T': card = 10; break;
        case 'J': card = 11; break;
        case 'Q': card = 12; break;
        case 'K': card = 13; break;
        }

        switch (suit) {
        case 'H': card |= HEART;    break;
        case 'D': card |= DIAMOND;  break;
        case 'C': card |= CLUB;     break;
        case 'S': card |= SPADE;    break;
        }

        return card;
}

/* This game has several piles of cards.  We represent each pile as a
   stack of several cards.  Each pile may hold at most a full deck
   (although this will never actually happen in the game).  "top" is
   an index to the entry *after* the top card.  Thus, if top == 0, the
   pile is empty. */
struct pile {
        int cards[52];
        int top;
};

/* The game has 13 piles.  We start counting at 1 so that we can use
   the value of the cards directly as indexes */
struct pile piles[14];

int card; /* the last card flipped */
int pile; /* the current pile */
unsigned exposed; /* the number of cards exposed */

/* We know that at the start of the game, each pile will have
   precisely four cards in it */
void init_piles (void)
{
        int i;
        for (i = 1; i <= 13; i++) {
                piles[i].top = 4;
        }
}

/* Remove a card from the indicated pile, and return it */
int pop (struct pile *pile)
{
        if (!pile->top) return 0;
        return pile->cards[--pile->top];
}

/* Input is given as a list of cards that form a deck, to be dealt.
   We skip the regular dealing and just read the cards in directly to
   the piles.  We call this "undealing" ;) */

/* Place the card read from a certain index in a certain row into the
   appropriate pile */
void undeal_card (int card, int row, int index)
{
        piles[index].cards[row] = card;
}

/* Read in an entire row of cards and place them appropriately */
```

```
void undeal_row (const char *s, int row)
{
        int i;
        char value, suit;
        int card;
        for (i = 13; i >= 1; i--) {
                value = *s++;
                suit = *s++;
                card = to_card (value, suit);
                s++;
                undeal_card (card, row, i);
        }
}

/* This is where the fun begins! :) */
int main (void)
{
        char line[256];

        while (1) {
                exposed = 0;
                pile = 13;
                init_piles ();

                fgets (line, sizeof line, stdin);
                if (line[0] == '#') break;

                /* Read and place all 4 rows of cards */
                undeal_row (line, 3);
                fgets (line, sizeof line, stdin);
                undeal_row (line, 2);
                fgets (line, sizeof line, stdin);
                undeal_row (line, 1);
                fgets (line, sizeof line, stdin);
                undeal_row (line, 0);

                /* Play the game */
                while (1) {
                        int lastcard = card;
                        card = pop (&piles[pile]);
                        if (!card) { card = lastcard; break; }
                        pile = card & VALUE_MASK;
                        exposed++;
                }
                printf ("%02d,%c%c\n", exposed, to_value (card),
                        to_suit (card));
        }
        return 0;
}
```

## String manipulation – embedded codes (C)

```
#include <stdio.h>
#include <string.h>
/* Dan Stutzbach - Fall 2001 */
/* Type of problem: simple string/pointer manipulation.  strchr() does most of work. */
int main (void)
{
        FILE *f;
        char line1[512], line2[512];
        f = fopen ("e.dat", "r");

        /* Repeat for each pair of data */
        while (1) {
                /* Algorithm:
                   1) Search for the first character of line1 (*s1) in line 2
                   2) Computer the interval from the start of line2 to
                   the first instance of *s1
                   3) Check to see if all of line1 can be matched
                   using that interval.  If so, we're done.
                   4) Otherwise, find the next instance of *s1 and try again.
                */

                char *s1, *s2,*base_s2, *finish;
                unsigned interval;

                fgets (line1, sizeof line1, f);
                if (line1[0] == '#') break;
                fgets (line2, sizeof line2, f);
                base_s2 = line2;

                /* Be sure to truncate the lines at the '*', which is
                   specified as the end-of-data marker */
                finish = strchr (line1, '*');   *finish = 0;
                finish = strchr (line2, '*');   *finish = 0;

        fail:
                /* This is where we return to if the computed interval
                   didn't work out. */
                /* Using "goto" in circumstances like this is much
                   faster than having to setup and use an extra flag
                   variable.  (Both for me to code, and for the
                   computer to execute) */

                s1 = line1;
                s2 = line2;
                base_s2 = strchr (base_s2+1, *s1);

                if (!base_s2) {
                        printf ("[%s] is not found.\n", line1);
                        continue;
                }

                interval = (base_s2 - s2) + 1;
                s2 = base_s2;

                while (*++s1) {
                        if (s2[interval] != *s1)
                                goto fail;
                        s2 += interval;
                }
                printf ("[%s] is found with encoding of %d.\n",line1,
interval);
        }
        return 0;
}
```

## GCD Algorithm (C)

```
/* Algorithm taken from The Art of Computer Programming, Volume 2, 3rd
   Edition, page 337.  More efficient and more generalized gcd
   algorithms may be found there.  This one is shortest to type :) */

/* Dan Ellsworth - Fall 2001 */

unsigned gcd (unsigned u, unsigned v)
{
        unsigned r;

        while (v) {
                r = u % v;
                u = v;
                v = r;
        }

        return u;
}

unsigned lcm (unsigned u, unsigned v)
{
        return u * v / gcd (u, v);
}

/* Example usage */
#include <stdio.h>

int main (void)
{
        int x = 75;
        int y = 250;

        printf ("gcd: %d, lcm: %d\n", gcd (x, y), lcm (x, y));

        return 0;
}
```

## Swampnet Routing – packet filtering rules (C)

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

/* Dan Stutzbach - Fall 2001 */

/* The given problem is to read in a set of packet filtering rules,
   and store them.  Next, read in a bunch of packets, and decide how
   many packets are dropped, how many are allowed, and how many rules
   are checked to find out.  */

/* This problem was fairly simple in theory, but took a long time to
   type in.  I made all the parsing functions advance the string as
   they went along, which saves some time reading the data in.
   Otherwise, I think it's a pretty straightfoward problem and
   implementation. */

/* For clarity */
typedef int bool;


/* They define a limit on the number of filter rules we'll be given,
   so we'll just use an array of structures.  We use unsigned values
   where it makes sense.  Some fields may have a special "match-all"
   value.  For these fields, we use a signed value and use -1 to
   represent "match-all".  Thankfully, none of these fields are
   already 32 bits. */

struct rule {
        signed protocol;
        unsigned src_ip;
        unsigned src_mask;
        unsigned dst_ip;
        unsigned dst_mask;
        signed src_port1;
        signed dst_port1;
        signed src_port2;
        signed dst_port2;
        bool deny;
};

/* Each packet has a similar structure.  We deliberately use the same
   names for the fields, as this will allow us to cut and paste code
   later more easily. */
struct packet {
        signed protocol;
        unsigned src_ip;
        unsigned dst_ip;
        signed src_port;
        signed dst_port;
};

/* We make it bigger than the 100 they specify Just In Case.  This
   will protect us from certain sorts of off-by-one errors they we (or
   the judges) might make.  */
struct rule rules[105];

/* This tells us how many entries in the "rules" global array are valid. */
```

```
unsigned num_rules = 0;

/* These are the values we need to compute and print out */
unsigned compared = 0;
unsigned permitted = 0;
unsigned denied = 0;

/* This is a simple little function that returns the next
   non-whitespace character in a string.  If the first character is
   non-whitespace, it does not advance the pointer at all. */
char *eatw (char *s)
{
        while (isspace ((int) *s)) s++;
        return s;
}

/* This returns an IP address, and advances the string pointer to just
   after the IP address.  */
unsigned get_ip (char **ip)
{
        unsigned num = 0;
        char *s = *ip;

        /* Remove whitespace */
        s = eatw (s);

        /* strtol() is great! It reads a number much like atoi(), but
           also advances the string pointer for us! */
        num |= strtol (s, &s, 10);

        /* Skip past the period in the IP address */
        s++;

        /* and repeat... */
        num |= strtol (s, &s, 10) << 8;
        s++;
        num |= strtol (s, &s, 10) << 16;
        s++;
        num |= strtol (s, &s, 10) << 24;

        /* Advance the string pointer in the caller */
        *ip = s;
        return num;
}

/* This returns a signed int, and advances the string pointer.  As a
   special case, it considers an input string of "*" to indicate
   match-all, and returns -1. */
signed get_int (char **ps)
{
        char *s = *ps;
        signed num;
        s = eatw (s);
        if (*s == '*') {
                s++;
                num = -1;
        }
        else num = strtol (s, &s, 10);
        *ps = s;
        return num;
}
```

```
/* This function checks to see if a packet matches a rule.  It returns
   true if it does, and false if it doesn't. */
bool test_rule (const struct packet *p, const struct rule *r)
{
        if ((p->src_ip & r->src_mask) != r->src_ip)
                return 0;
        if ((p->dst_ip & r->dst_mask) != r->dst_ip)
                return 0;
        if ((r->src_port1 != -1) && ((p->src_port < r->src_port1)
                                     || (p->src_port > r->src_port2)))
                return 0;
        if ((r->dst_port1 != -1) && ((p->dst_port < r->dst_port1)
                                     || (p->dst_port > r->dst_port2)))
                return 0;
        if ((r->protocol != -1) && (r->protocol != p->protocol))
                return 0;
        return 1;
}

/* This function checks to see if the packet matches any of the rules.
   It tries each rule sequentially until a match is found.  When a
   match is found, it checks the "deny" field of the rule, and
   increments "denied" or "permitted" as appropriate.  It also
   increments "compared" for each rule checked. */
void test_list (const struct packet *p)
{
        unsigned start;
        for (start = 0; start < num_rules; start++) {
                compared++;
                if (test_rule (p, &rules[start])) {
                        if (rules[start].deny)
                                denied++;
                        else permitted++;
                        break;
                }
        }

}

int main (void)
{
        char line[512];
        FILE *f;

        f = fopen ("h.dat", "r");

        /* Repeat for each set of test data... */
        while (1) {
                num_rules = 0;
                compared = 0;
                permitted = 0;
                denied = 0;

                /* Repeat for each rule */
                while (1) {
                        struct rule *rule = &rules[num_rules];
                        char *s;
                        if (!fgets (line, sizeof line, f)) break;
                        s = line;
                        s = eatw (s);
                        if (!*s) break;
                        num_rules++;
                        rule->protocol = get_int (&s);
                        rule->src_ip   = get_ip (&s);
                        rule->src_mask = get_ip (&s);
                        rule->src_port1 = get_int (&s);
                        if (*s == '0') {
                                s++;
                                rule->src_port2 = get_int (&s);
                                if (rule->src_port2 < rule->src_port1) {
                                        rule->src_port1 ^= rule->src_port2;
                                        rule->src_port2 ^= rule->src_port1;
                                        rule->src_port1 ^= rule->src_port2;
                                }
                        } else rule->src_port2 = rule->src_port1;
                        rule->dst_ip   = get_ip (&s);
                        rule->dst_mask = get_ip (&s);
                        rule->dst_port1 = get_int (&s);
                        if (*s == '0') {
                                s++;
                                rule->dst_port2 = get_int (&s);
                                if (rule->dst_port2 < rule->dst_port1) {
                                        rule->dst_port1 ^= rule->dst_port2;
                                        rule->dst_port2 ^= rule->dst_port1;
                                        rule->dst_port1 ^= rule->dst_port2;
                                }
                        } else rule->dst_port2 = rule->dst_port1;
                        s = eatw (s);
                        rule->deny = !strncasecmp (s, "deny", 4);
                }

                if (!num_rules) break;

                /* Repeat or each packet... */
                while (1) {
                        struct packet packet;
                        char *s;
                        if (!fgets (line, sizeof line, f)) break;
                        s = line;
                        s = eatw (s);
                        if (!*s) break;
                        packet.src_ip   = get_ip (&s);
                        packet.src_port = get_int (&s);
                        packet.dst_ip   = get_ip (&s);
                        packet.dst_port = get_int (&s);
                        packet.protocol = get_int (&s);

                        test_list (&packet);
                }

                printf ("%d %d %d\n", compared, permitted, denied);
        }

        return 0;
}
```

## Postscript Emulation – coordinate transformation   (C)

```
/* @JUDGE_ID: 14703TH 329 C */
/* Tim Singer - Fall 2001 */
/* Postscript emulation */
/*  compile with '-lm' to test */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

/* coordinate transform matrix (rotate,scale,translate) */
/*  used in absolute calculations */
double T[3][3] = {
       { 1, 0, 0 },
       { 0, 1, 0 },
       { 0, 0, 1 }
};

/* direction vector transform matrix (rotate,scale) */
/*  used in relative calculations */
double V[3][3] = {
       { 1, 0, 0 },
       { 0, 1, 0 },
       { 0, 0, 1 }
};

/* multiply two 3x3 matrices (rows * columns) */
void matmul(double A[3][3],double B[3][3]) {
       double temp[3][3];
       int i,j;
       for(i=0; i<3; i++) for(j=0; j<3; j++) {
              temp[i][j]=A[i][0]*B[0][j];
              temp[i][j]+=A[i][1]*B[1][j];
              temp[i][j]+=A[i][2]*B[2][j];
       }
       for(i=0; i<3; i++) for(j=0; j<3; j++)
              A[i][j]=temp[i][j];
       return;
}

/* given absolute coords, print transformed coordinates */
void transformcoords(double a, double b) {
       double temp[3];
       int i;

       for(i=0; i<3; i++) {
              temp[i]=T[i][0]*a;
              temp[i]+=T[i][1]*b;
              temp[i]+=T[i][2];
       }

      /* divide by homogeneous coordinates */
       temp[0]/=temp[2]; if(temp[0]<0.000001 && temp[0]>-0.000001) temp[0]=0;
       temp[1]/=temp[2]; if(temp[1]<0.000001 && temp[1]>-0.000001) temp[1]=0;
       printf("%.6g %.6g ",(float)temp[0],(float)temp[1]);
       return;
}

/* given relative coords, print transformed coordinates */
void transformvect(double a, double b) {
       double temp[3];
       int i;

       for(i=0; i<3; i++) {
         temp[i]=V[i][0]*a;
```

```
              temp[i]+=V[i][1]*b;
              temp[i]+=V[i][2];
       }
        /* divide by homogeneous coordinates */
       temp[0]/=temp[2]; if(temp[0]<0.000001 && temp[0]>-0.000001) temp[0]=0;
       temp[1]/=temp[2]; if(temp[1]<0.000001 && temp[1]>-0.000001) temp[1]=0;
       printf("%.6g %.6g ",temp[0],temp[1]);
       return;
}

/* uppercase -> lowercase */
void tolow(char* c) {
  for(;*c;c++) if(*c>='A' && *c<='Z') *c+=' ';
}

/* postscript parser */
int main() {
       char cmd[256];
       double arg1,arg2;
       double M[3][3];
       scanf("%s",cmd);
       while(strcmp(cmd,"*")) {
          /* Parse the command */
              arg1=atof(cmd);
              scanf("%s",cmd);
              tolow(cmd);
              if(!strncmp(cmd,"ro",2)) {
                 /* Rotate coordsys arg1 degrees counterclockwise */
                 /* Convert arg1 from degrees to radians */
                     arg1/=180;
/* cclock rotation: */      arg1*=3.1415926535897932;
/* {{ cos,-sin,0 }  */      M[0][0]=cos(arg1); M[0][1]=-sin(arg1); M[0][2]=0;
/*  { sin, cos,0 }  */      M[1][0]=sin(arg1); M[1][1]=cos(arg1);  M[1][2]=0;
/*  { 0,   0,  1 }} */      M[2][0]=0;         M[2][1]=0;          M[2][2]=1;
                     matmul(T,M);
                     matmul(V,M);
              } else {
                     arg2=atof(cmd);
                     scanf("%s",cmd);
                     tolow(cmd);
                     if(!strncmp(cmd,"tr",2)) {
/* translate x,y: */    /* Translate coordsys (arg1,arg2) */
/* {{ 1, 0, x }  */         M[0][0]=1;    M[0][1]=0;    M[0][2]=arg1;
/*  { 0, 1, y }  */         M[1][0]=0;    M[1][1]=1;    M[1][2]=arg2;
/*  { 0, 0, 1 }} */         M[2][0]=0;    M[2][1]=0;    M[2][2]=1;
                            matmul(T,M);
                     } else if(!strncmp(cmd,"sc",2)) {
/* scale x,y: */        /* Scale coordsys (arg1,arg2) */
/* {{ x, 0, 0 }      */           M[0][0]=arg1; M[0][1]=0;    M[0][2]=0;
/*  { 0, y, 0 }  */         M[1][0]=0;    M[1][1]=arg2; M[1][2]=0;
/*  { 0, 0, 1 }} */         M[2][0]=0;    M[2][1]=0;    M[2][2]=1;
                            matmul(T,M);
                            matmul(V,M);
                     } else if(cmd[0]!='r') {   /* not relative */
                            transformcoords(arg1,arg2);
                            printf("%s\n",cmd);
                     } else {                   /* relative */
                            transformvect(arg1,arg2);
                            printf("%s\n",cmd);
                     }
              }
           scanf("%s",cmd);
       }
       return 1;
}
```

## Base addition – solve equation for base (C)

```
#include <stdio.h>
#include <string.h>

/* Tim Singer - Fall 2001 */

/* base arithmetic - addition using carry on
   arrays of ints representing digits*/

/* strings for holding base digits */
char s1[81], s2[81], s3[81];

/* ints for digit conversion */
int i1[81], i2[81], i3[81], isum[81];

/* convert an array of chars in base b to an array of ints
   representing the individual digits */
int convert(char* s, int* i, int b) {
  int l = 81-strlen(s);
  for(int j=0; s[j]; j++) {
    i[j+l] = s[j] > '9' ? s[j]-'A'+10 : s [j] - '0';
    if(i[j+l] >= b) return 0;

  }
  for (int k=0; k<l; k++) i[k]=0;
  return 1;
}

/* check for equality between two arrays of digits */
int equal(int* x, int* y) {
  for(int i=0; i<81; i++) if(x[i] != y[i]) return 0;
  return 1;
}

/* add array x to array y and store in array z (working in base b) */
void add(int* x, int* y, int* z, int b) {
  int c=0;
  for(int i=80; i>=0; i--) {
    z[i] = x[i] + y[i] + c; c=0; /* add with carry and clear carry */
    if(z[i]>=b) {                /* set carry on overflow */
      z[i] -= b;
      c = 1;
    }
  }
}

int main(void) {
  while(1) {
    int b=0;
   /* read in the three numbers */
    scanf("%s",s1); if(!strcmp(s1,"0")) break;
    scanf("%s",s2); scanf("%s",s3);

   /* for each base (up to 36): */
    for(b = 2; b <= 36; b++) {
     /* convert the numbers */
      if(!convert(s1,i1,b)) continue;
      if(!convert(s2,i2,b)) continue;
      if(!convert(s3,i3,b)) continue;
     /* add the numbers */
      add(i1,i2,isum,b);

     /* print and break out if equal */
      if(equal(i3,isum)) {
        printf("%s + %s = %s in base %i\n",s1,s2,s3,b);
        break;
      }
    }
   /* print failure if all bases checked */
    if(b==37) printf("%s + %s != %s\n",s1,s2,s3);
  }
  return 1;
}
```

## Folding dot – geometry (C++)

```
#include <iostream.h>

/* Dan Ellsworth - Fall 2001 */

// Nothing resursive, so globals are fairly safe
int pages_on_top;
int pages_under;
double page_x;
double page_y;
double dot_x;
double dot_y;
int fold_change;
char on_top;

// func to make the changes associated with folding the page
void foldpage(char a, char b);
// func to make the output a little easier
void output();

int main() {
   int count=1;
   while(cin) {
      pages_on_top=0;
      pages_under=0;
      fold_change=1;
      cin >> page_x >> page_y;
      cin >> dot_x >> dot_y >> on_top;
      char fold;
      char dir;
      char control;
      cin >> control;

      // kludge to capture an error with improper loop termination
      // Perhaps due to a compiler optimzation?
      if(control=='S') {
         return 1;
      }

      cout << "Paper number " << count << endl;
      cout << "Beginning paper ";
      output();
      while(control!='S') {
         fold=control;
         cin >> dir;
         foldpage(fold,dir);
         cin >> control;
      }
      cout << "After folding paper. Paper ";
      output();
      count++;
   }
   return 1;
}

void output() {
   cout.setf(ios::fixed);
   cout.precision(6);
   cout.setf(ios::showpoint);
   cout << "dimensions " << page_x << " X " << page_y << endl;
   cout << "Dot is on ";
   if(on_top=='T') {
      cout << "TOP";
   } else {
      cout << "BOTTOM";
   }
   cout << " of page " << pages_on_top+1 << ". Position: " << dot_x << " X " << dot_y <<
endl << endl;
}

void foldpage(char fold,char dir) {
   int top_change=0;
   // Set toppedness of the dot and change paper dimensions
   if(fold=='T' && dot_y>page_y/2) {
      page_y=page_y/2;
      dot_y=page_y-dot_y;
   } else if (fold=='B' && dot_y<page_y/2) {
       dot_y=page_y-dot_y;
       page_y=page_y/2;
   } else if (fold=='T' || fold=='B') {
       page_y=page_y/2;
       top_change=1;
   }
   if(fold=='L' && dot_x >page_x/2) {
      page_x=page_x/2;
      dot_x=page_x-dot_x;
   } else if (fold=='R' && dot_x<page_x/2) {
      dot_x=page_x-dot_x;
      page_x=page_x/2;
   } else if( fold=='R' || fold=='L') {
      top_change=1;
      page_x=page_x/2;
   }

   // if the top changes make some changes to the accounting for how many
   // pages are above and below the dot
   if(top_change==1) {
      int t=pages_on_top;
      pages_on_top=pages_under;
      pages_under=t;
      if(dir=='U') {
         dir='O';
      } else {
         dir='U';
      }
      if (on_top='T') {
         on_top='B';
      } else {
         on_top='T';
      }
   }
   if(dir=='O') {
      pages_on_top+=fold_change;
   } else {
      pages_under+=fold_change;;
   }

   //update how many layer happen with the next fold
   fold_change*=2;

   //make the coordinate values positive
   if(dot_x<0) {
      dot_x*=-1;
   }
   if(dot_y<0) {
      dot_y*=-1;
   }
}
```

## Burgers – combinations and probability (C++)

```
#include <iostream.h>
/* Tim Singer - Fall 2001 */
//Hamburger probability problem
// Given an equal number of hamburgers and cheeseburgers and as many guests as there
// are burgers, what is the probability of the last two people getting the same type
// of burger?
//  Answer (N=#guests):
//  __N-2   /N-2\
//  \       \ i /
//  /_     -------
//  i=N/2    (N-2)
//          2

double pow(int a, int n) { //power function - compute a^n
   double x=a;
   for (int m = n-1; m>0; m--) x *= (double)a;
   return x;
}

//choose function: (n choose c) = (n!)/((c!)(n-c)!)
double choose(int n, int c) {
    double x = n;
    int m;
    for (m = n-1; m>n-c; m--) {
        x *= (double)m;                    //this line computes n!/c!
        if(n-m <= c) x /= (double)(n-m);   //this line partially computes
    }                                      // 1/((n-c)!)

   //this line finishes the 1/((n-c)!) computation
   //  - it is split to keep the numbers from growing too huge.
    for (; m>=n-c; m--) if(n-m <= c) x /= (double)(n-m);
    return x;
}

//probability of n heads in g tries
// = (#of n-head arrangements) / (#of total arrangements) = (g choose n)/(2^g)
double prob(int n, int g) {
    double x = choose(g,n);
    x /= pow(2,g);
    return x;
}

//Calculate probability of *at least* nguests/2 heads in nguests-2 tries and sum
void calculateprob(int nguests) {
     double total=0;
     for(int nheads=nguests/2; nheads<nguests-1; nheads++) {
         total+=prob(nheads,nguests-2);
     }
     cout << (float)total*2 << endl;
}

void main(void) {
  //read number of entries
   int n;  cin >> n;
  //read entries
   int *e = new int[n];
   for(int i=0; i<n; i++) {
      cin >> e[i];
   }
  //calculate probabilities
   for(int j=0; j<n; j++) {
       calculateprob(e[j]);
   }
}
```

## Level ordering of weighted tree    (C++)

```
/*

This program takes in a data set of the following form. The first line
contains a number corrasponding to the number of nodes in the tree.
Each line there after contains a one word name followed by a space followed
by the wieght of the node.

Example Input:
5
Eugene 10
Medford 20
Newport 1
Portland 5
Salem 10

The output is a level ordered output of the resulting tree.
*/
/* Dan Ellsworth - Fall 2001 */

#include <iostream.h>

/*******************************************************
       Some structures I thought might be useful
 *******************************************************/
struct nameWeight {
   char *name_ptr; // Pointer to the item name
   int freq;     // The frequency this node will be looked for
};

struct tableNode {
   int root;     // Index of the root within the list
   int cost;     // cost of this node
   int weight;   // weight of this node
};

/*******************************************************
               The simple queue class
*******************************************************/
class queue {
   private:
      int *queue_ptr;
      int length;
      int current;
      int end;
   public:
      int* fetch();
      void insert(int start, int end);
      void init(int length);
};

/****** initalizer for the queue ******/
void queue::init(int len) {
   length=2*len;
   queue_ptr=new int[length];
   current=0;
   end=0;
}

/****** this is how I insert data ******/
inline void queue::insert(int x, int y) {
   queue_ptr[end++]=x;
   queue_ptr[end++]=y;
}

/****** this is how I get data back out ******/
```

```
int* queue::fetch() {
   if (current==end) { return 0; }
   current+=2;
   return queue_ptr+(current-2);
}


/*****************************************************
                  The table class
*****************************************************/

class table {
   private:
      tableNode *table_ptr; // a pointer to the first element
      int dim;              // dimension of the table
   public:
      void init(int dim);
      tableNode fetch(int x, int y);
      void insert(int x, int y, int root, int weight, int cost);
};

/****** initalizer for the table ******/
void table::init(int x) {
   dim=x;
   table_ptr=new tableNode[dim*dim];
}

/****** this is how I insert elements ******/
void table::insert(int x, int y, int root, int weight, int cost) {
   int tmp=x*dim+y;
   table_ptr[tmp].root=root;
   table_ptr[tmp].cost=cost;
   table_ptr[tmp].weight=weight;
}

/****** this is how I return elements ******/
inline tableNode table::fetch(int x, int y) {
   return table_ptr[x*dim+y];
}

/*****************************************************
    Prototypes for function bodies soon to come...
*****************************************************/
void readInputs(nameWeight elements[],int numberOfEntries);
void optimalTree(nameWeight elements[], int start, int end, table theTable);
void writeTree(nameWeight elements[],queue printQueue,table theTable);

/*****************************************************
           Here starts the main method...
*****************************************************/
int main() {

   // Read in the number of elements to put in the tree
   int numElements;
   cin >> numElements;
   nameWeight *elements; // an array of the elements we need to have in the BST
   elements= new nameWeight[numElements]; // allocate the memory for the heap so
                                          // that it doesn't dissapear

   // initalize the table object... no real reason to do it this early other than
   //    it seemed like a good idea when I put it in here.
   table theTable;
   theTable.init(numElements);

   // Read in all of the elemenets
   readInputs(elements,numElements);

   // Start the recursion to build all of the trees
   //    This call here fills in the table
   optimalTree(elements,numElements,numElements,theTable);

   // Output the tree
   //   the writeTree function has all of the logic for properly tracing back through the
table
   queue printOrder;
   printOrder.init(numElements);
   printOrder.insert(0,numElements-1);  // setup the queue so that output can meet the
spec
   writeTree(elements,printOrder,theTable);
   return 0;
}

/*******************************************************
             -- writeTree --

 This function takes in:
    nameWeight[] -- contains the sorted list of elements and weights for the BST
    queue        -- this is a queue to facilitate printing the node in level order rather
                    than pre- or post-order
    table        -- a table object that contains all of the needed data to construct the
BST
 This function returns:
    Spools the contents of the optimal BST represented by the table to stdout
*******************************************************/

void writeTree(nameWeight elements[],queue printOrder,table theTable) {
   int *int_ptr=printOrder.fetch();
   if (int_ptr==0) { // end of queue reached
      return;
   }
   int tmp=theTable.fetch(*int_ptr,*(int_ptr+1)).root;

   // if this node has no childern....
   if (int_ptr[0]==int_ptr[1]) {
      cout << elements[tmp].name_ptr << " (0,0)\n";
      writeTree(elements,printOrder,theTable);
      return;
   }

   // fetch the roots of the right and left childern
   int left=theTable.fetch(*int_ptr,tmp-1).root;
   int right=theTable.fetch(tmp+1,*(int_ptr+1)).root;

   cout << elements[tmp].name_ptr << " (";

   if(int_ptr[0]>tmp-1) { // if there is no left child...
      cout << "0,";
   } else {          // if there is a left child...
      cout << elements[left].name_ptr << ',';
   }
   if(tmp+1>int_ptr[1]) { // if there is no right child...
      cout << "0)\n";
   } else {        // if there is a right child...
      cout << elements[right].name_ptr << ")\n";
   }
   if(int_ptr[0]<=tmp-1) { // if there is a left child, add it to the queue
      printOrder.insert(int_ptr[0],tmp-1);
   }
   if(tmp+1<=int_ptr[1]) { // if there is a right child, add it to the queue
      printOrder.insert(tmp+1,int_ptr[1]);
   }
   writeTree(elements,printOrder,theTable);
}

/*******************************************************
                   -- optimalTree --
 This function takes in:
    nameWeight[] -- contains the sorted list of elements and weights for the BST
    int len      -- the length of the segment to compute
    int end      -- index of the last node in the list
    table        -- a table object that holds all of the needed data to construct the BST
 This function returns:
    the table is filled in to represent the optimal BST for the set of elements
*****************************************************/
void optimalTree(nameWeight elements[], int len, int end, table theTable) {

   if(len==1) { // base case for the recursion
      for(int i=0;i<end;i++) { // fill in the diagonal
         theTable.insert(i,i,i,elements[i].freq,0);
      }
      for(int i=0;i<end-1;i++) { // fill in the second diagonal
         tableNode left=theTable.fetch(i,i);
         tableNode right=theTable.fetch(i+1,i+1);
         if(left.weight*2-left.cost+right.weight<right.weight*2-right.cost+left.weight) {
            theTable.insert(i,i+1,i+1,left.weight*2-left.cost+right.weight,left.weight);
         } else {
            theTable.insert(i,i+1,i,left.weight+2*right.weight-right.cost,right.weight);
         }
      }
      return; // base case complete, returning to next caller on the stack
   }

   optimalTree(elements,len-1,end,theTable); // fill in the needed data in the table for
this iteration

   for(int i=0;i<end-len;i++) { // loop through all segments of length len

      // initalize values for this segment, testing the first case...
      int minSoFar=theTable.fetch(i,i).weight+2*theTable.fetch(i+1,i+len).weight-
theTable.fetch(i+1,i+len).cost;
      int minCost=theTable.fetch(i+1,i+len).weight;
      int minRoot=i;

      for(int j=i+1;j<i+len;j++) { // loop through possible roots for this segment,
testing for more optimal values
         int tmp=theTable.fetch(j,j).weight;
         int tmp2=theTable.fetch(i,j-1).weight*2-theTable.fetch(i,j-
1).cost+tmp+2*theTable.fetch(j+1,i+len).weight-theTable.fetch(j+1,i+len).cost;
         if (tmp2<minSoFar) { // more optimal value found, record and continue with
testing
            minSoFar=tmp2;
            minRoot=j;
            minCost=theTable.fetch(i,j-1).weight+theTable.fetch(j+1,i+len).weight;
         }
      }
      int tmp=theTable.fetch(i+len,i+len).weight;
      if(2*theTable.fetch(i,i+len-1).weight-theTable.fetch(i,i+len-1).cost+tmp<minSoFar )
{ // test last case
         minSoFar=2*theTable.fetch(i,i+len-1).weight-theTable.fetch(i,i+len-1).cost+tmp;
         minRoot=i+len;
         minCost=theTable.fetch(i,i+len-1).weight;
      }
      theTable.insert(i,i+len,minRoot,minSoFar,minCost); // optimal value for this segment
is recorded in the table
   }
}

/*****************************************************
                   -- readInputs --
 This function takes in:
    nameWeight[] -- contains the sorted list of elements and weights for the BST
    int entries  -- the number of elements and weights to read in
 This function returns:
    this function reads input from stdin and places it into an array for later reference
*****************************************************/
void readInputs(nameWeight elements[],int entries) {
   char tmpName[100]; // a temporary place to put names
   int tmpFreq;       // a temporary place to put frequencies

   int i=0;
   for(;i<entries;i++) {

      cin >> tmpName >> tmpFreq;

      int j=0;
      while(tmpName[j++]!='\0') { }


      char *name; // char space just the length of the string we're interested in
      name=new char[j]; // get some space from the heap so that the string doesn't
dissapear when this function returns

      // copy the string
      for(int k=0;k<j;k++) {
         name[k]=tmpName[k];
      }

      // add the entry to the list
      elements[i].name_ptr=name;
      elements[i].freq=tmpFreq;
   }
}
```

## Determine if directed graph is a tree   (C++)

```
#include <iostream.h>
/* Tim Singer - Fall 2001 */
// is it a tree?
//  1. find a source (no edges coming in)
//  2. recursively traverse all paths from the source
//      - if same node is encountered twice, it is not a tree.
//  3. If all nodes were encountered, it is a tree.

int numedges;
int orig[1000];
int dest[1000];
int nodenum[2000];
int numnodes;

//finds node #num in an array of dimension n - TRUE if succeeds
int findnode(int num, int* arr, int n) {
   for(int i=0; i<n; i++) if(arr[i]==num) return 1;
   return 0;
}

//finds a source and returns the node#
int findroot() {
   for(int i=0; i<numedges; i++) {
     int cur = orig[i];
     if(!findnode(cur,dest,numedges)) return cur;
   }
   return -1;
}

//traverses from this node - FALSE if a node was encountered twice
int traverse(int root) {
    if(findnode(root,nodenum,numnodes)) return 0;
    nodenum[numnodes]=root;
    numnodes++;
    for(int i=0; i<numedges; i++) {
        if(orig[i]==root) {
            if(!traverse(dest[i])) return 0;
        }
    }
    return 1;
}

//TRUE if all nodes in the edge list were found
int allnodesfound() {
   for(int i=0; i<numedges; i++) {
       if(!findnode(orig[i],nodenum,numnodes) ||
          !findnode(dest[i],nodenum,numnodes))
            return 0;
   }
   return 1;
}

int main() {
   int n1,n2;
   cin >> n1 >> n2;
   int Case=0;

   while(n1+n2>=0) {  //input values positive if valid
```

```
    //input the edges
     numedges=0;
     while(n1+n2!=0) {     //input values nonzero if valid
       orig[numedges]=n1;
       dest[numedges]=n2;
       numedges++;
       cin >> n1 >> n2;   //read in the next values (for while loop)
     }
     cin >> n1 >> n2;     //read in the next values (for while loop)

     Case++;

    //No edges? then it is a tree, albeit an empty one.
     if(numedges==0) {
       cout << "Case " << Case << " is a tree." << endl;
       continue;
     }

    //mark all nodes untaken
     for(int i=0; i<numedges; i++) nodenum[i]=0;

    //find the root
     int n=findroot();

    //No source? not a tree.
     if(n<0) {
       cout << "Case " << Case << " is not a tree." << endl;
       continue;
     }

    //Traverse the tree and check for coverage of all nodes
     if(traverse(n) && allnodesfound()) {
       cout << "Case " << Case << " is a tree." << endl;
     } else {
       cout << "Case " << Case << " is not a tree." << endl;
     }
   }
}
```

## Anagram – recursive enumeration of permutations (Java)

```
/*
 * 195 Anagram
 *
 * A simple recursive enumeration of distinct permutations from the input.
 */

/* Carl Howells - Fall 2001 */

import java.util.*;
import java.io.*;

class Main
{
    public static void main(String [] args) throws Exception{
        int times = Integer.parseInt(readline());

        for (int j = 0; j < times; j++) {
            String s = readline();


            // this big mess is just to sort alphabetically

            char [] c = s.toCharArray();

            Character [] C = new Character[c.length];
            for (int i = 0; i < C.length; i++)
                C[i] = new Character(c[i]);

            Arrays.sort(C, new Comparator()
            {
                public int compare(Object obj1, Object obj2)
                {
                    Character o1 = (Character)obj1;
                    Character o2 = (Character)obj2;

                    int a = Character.toLowerCase(o1.charValue());
                    int b = Character.toLowerCase(o2.charValue());

                    if (a < b) return -1;
                    if (a > b) return 1;

                    if (o1.charValue() < o2.charValue()) return -1;
                    if (o1.charValue() > o2.charValue()) return 1;

                    return 0;
                } // compare

            });

            for (int i = 0; i < c.length; i++)
                c[i] = C[i].charValue();

            s = new String(c);


            // this is the interesting permutation bit

            permute(s);
        }
    }

    static void permute(String s) {
        recPermute(s, "");
    }
    static void recPermute(String s1, String s2) {

        // base case...  The first string is empty, so print the second
        if (s1.equals("")) {
            System.out.println(s2);
        }

        // iterate through the characters, moving each in turn to back
        // of the second list.  They are moved to the back to preserve
        // ordering of the incoming string.

        for (int i = 0; i < s1.length(); i++) {

            char c = s1.charAt(i);

            // this test that prevents recursion on redundant cases.
            // it works by checking to see if it has already put the
            // character it's currently on at the end of the String it's
            // building, and if so, doesn't continue down this branch.
            if (i == s1.indexOf(c)) {
                String s3 = s1.substring(0, i) + s1.substring(i + 1);
                String s4 = s2 + c;
                recPermute(s3, s4);
            }
        }


    }

    // IO Stuff..

    static BufferedReader r = new BufferedReader(new InputStreamReader(System.in));

    static String readline() throws Exception {
        return r.readLine();
    }
}
```

## Doors – line intersection and Floyd-Warshall algorithm (Java)

```
/*
 * Solution to problem 393 - Walls
 * Uses a line segment intersection test and the Floyd-Warshall algorithm
 * to solve the problem. Unfortunately, the line intersection test isn't very general,
 * as it only needed to work when one of the segments was vertical.
 */
/* Carl Howells - Fall 2001 */
import java.util.*;
import java.text.*;
import java.io.StreamTokenizer;
import java.io.InputStreamReader;

class Main
{
    // More of Carl's standard IO stuff...
    static StreamTokenizer tok = new StreamTokenizer(new InputStreamReader(System.in));

    static int readInt() throws Exception {
        tok.nextToken();
        return (int)tok.nval;
    }

    static double readDouble() throws Exception {
        tok.nextToken();
        return tok.nval;
    }

    public static void main(String [] args) throws Exception  {
        while (true) {
            int size = readInt();
            if (size < 0) break;

            // read the wall data into an array...  The first dimension
            // is the wall index.  The second dimension is the position
            // information.  Position 0 is the x coordinate, and positions
            // 1 through 4 are the end points of the wall along the y axis
            double [][] walls = new double[size][5];
            for (int i = 0; i < size; i++)
                for (int j = 0; j < 5; j++)
                    walls[i][j] = readDouble();

            // see the descriptions of these functions
            double [][] distances = calcdist(walls);
            double result = shortest(distances);

            // I found java's kind of equivalent to printf for formatting floats.
            NumberFormat nf = new DecimalFormat("#00.00");
            System.out.println(nf.format(result));
        }
    }

    static double [][] calcdist(double [][] walls) {
        // this method creates an array containing distances from each of the important
        // positions to each of the other positions.  The important positions are wall
        // end points and the start and goal locations.
        int size = 4 * walls.length + 2;
        double [][] result = new double[size][size];

        // for simplicity, create a new array to put the all the important locations in
        double [][] pos = new double[size][2];
        pos[0][0] = 0.0; pos[0][1] = 5.0; pos[1][0] = 10.0; pos[1][1] = 5.0;
        int count = 2;
        for (int i = 0; i < walls.length; i++){
            for (int j = 1; j < 5; j++){
                pos[count][0] = walls[i][0];
                pos[count][1] = walls[i][j];
                count++;
            }
        }

        // now, fill in the result array.  For this pass, The range is being set
        // to Double.MAX_VALUE if there is no direct path between the two points
        for (int i = 0; i < size; i++)
        {
            for (int j = i + 1; j < size; j++)
            {
                if (intersect(pos[i][0], pos[i][1], pos[j][0], pos[j][1], walls))
                {
                    result[i][j] = result[j][i] = Double.MAX_VALUE;
                }
                else
                {
                    double dx = pos[i][0] - pos[j][0];
                    double dy = pos[i][1] - pos[j][1];
                    result[i][j] = result[j][i] = Math.sqrt(dx * dx + dy * dy);
                }
            }
        }
        return result;
    }

    static boolean intersect(double x1, double y1, double x2, double y2, double [][]
walls)  {
        // this is a rather rigid brute force solution.  It just checks each wall
        // to see if it intersects with the segment between the given coordinates.
        double slope = (y1 - y2) / (x1 - x2);
        for (int i = 0; i < walls.length; i++){

            double x = walls[i][0];
            double y = y1 + slope * (x - x1);

            // if outside the segment, check the next wall
            if (x <= Math.min(x1, x2) || x >= Math.max(x1, x2)) continue;

            // if intersects with the current wall, return true
            if (y < walls[i][1]) return true;
            if (walls[i][2] < y && y < walls[i][3]) return true;
            if (walls[i][4] < y) return true;
        }
        // didn't intersect with anything
        return false;
    }

    static double shortest(double [][] A) {
        // this is just the floyd warshall algorithm, which I lean on
        // so often because it is simple to write.
        for (int i = 0; i < A.length; i++){
            for (int j = 0; j < A.length; j++){
                for (int k = 0; k < A.length; k++){
                    if (A[i][k] + A[k][j] < A[i][j]) A[i][j] = A[i][k] + A[k][j];
                }
            }
        }
        // return value is distance between points 0 and 1, which are start and end point
        return A[0][1];
    }
}
```

## Determine if directed graph is a tree   (Java)

```
/* Zebin Chen - Fall 2001 */
import java.io.*;
import java.util.*;

public class Tree2 {
  public static boolean debug=false;

  public Tree2() {
  }
  public static void main(String[] args) throws Exception {
      if (args.length>0 && args[0].equals("-debug")) {
          debug=true;
      }
      Vector v=new Vector();
      Vector tv=new Vector();
      BufferedReader d = new BufferedReader(new InputStreamReader (System.in));
      String temp, a, b;
      StringTokenizer st;
      Vector output=new Vector();
      Hashtable ht=new Hashtable();

      /**
       * Read input data and save them to buffer for future processing - separate I/O
       *    and tree decision.
       */
      do {
          temp=d.readLine();
          if (temp==null)
              break;
          temp=temp.trim();
          if (temp.startsWith("-1"))
              break;
          if (temp.equals(""))
              continue;
          st=new StringTokenizer(temp);
          while (st.hasMoreTokens()) {
              a=st.nextToken();
              b=st.nextToken();
              if (a.equals("0") && b.equals("0")) {
                  v.addElement(tv);
                  tv=new Vector();
                  break;
              } else {
                  tv.addElement(a);
                  tv.addElement(b);
              }
          }
      } while (true);

      /**
       * A tree can be fully decided by the following condition:
       *   1. every node can go up to look for oldest ancestor; if all nodes have same
       *   oldest ancestors (root), is a tree. Otherwise is not a tree.
       *   2. every node has at most one father. Specially, for root, no father; for
       *   internal node as well as leaf, one father.
       */
      for (int i=0; i<v.size(); i++) {
          tv=(Vector)v.elementAt(i);
          boolean ifTree=true;
          ht=new Hashtable();
          for (int j=0; j<tv.size();) {
              a=(String)tv.elementAt(j);
              b=(String)tv.elementAt(j+1);
              j+=2;
              TreeNode tn=(TreeNode)ht.get(b);
              /**
               * if a node have more than one father, NOT a tree.
               */
              if (tn!=null) {
                  ifTree=false;
                  break;
              } else {
                  ht.put(b, new TreeNode(b, a));
              }
          }
          Enumeration enum=ht.keys();
          String key0=(String)enum.nextElement();
          String root0=getRootKey(ht, key0);
          while (enum.hasMoreElements()) {
              String key1=(String)enum.nextElement();
              String root1=getRootKey(ht, key1);
              /**
               * if a node have more than one root, NOT a tree.
               */
              if (!root0.equals(root1)) {
                  ifTree=false;
                  if (debug)
                      System.out.println("dupliacate root: " + key0 + " " + key1);
                  break;
              }
          }
          if (ifTree)
              output.addElement(" ");
          else
              output.addElement(" not ");
      }
      for (int i=0; i<output.size(); i++) {
          System.out.println("Case " + (i+1) + " is" + output.elementAt(i) + "a tree.");
      }
  }
  public static String getRootKey(Hashtable ht, String key) {
      if (debug)
          System.out.print("key=" + key + "; ");
      String value=key;
      Object tn=ht.get(value);
      while (tn!=null) {
          value=((TreeNode)tn).father;
          tn=ht.get(value);
      }
      if (debug)
          System.out.println("root is " + value);
      return value;
  }
}

class TreeNode {
  public String key, father;
  public Vector children;
  private String NAK="-1";
  public TreeNode(String key, String father) {
    this.key=key;
    this.father=father;
    this.children=new Vector();
  }
  public TreeNode(String key) {
    this.key=key;
    this.father=NAK;
    this.children=new Vector();
  }
}
```

## Clock patience – card game   (Java)

```
/* Arel Cordero */
// CLOCK PATIENCE

// Play a card game where input is a shuffled deck of cards
// Cards are represented in input as two char strings like "5H"

// Problem 170 from University of Valladolid (SPAIN) archive


import java.io.*;
import java.util.*;

class a {

    public static void main (String[] args) throws Exception {

      // Read from file
      BufferedReader file = new BufferedReader (new FileReader ("a.dat"));


      /*********************** or Read from standard IN ***************
      BufferedReader file =
           new BufferedReader (new InputStreamReader (System.in));
      ****************************************************************/

      /********* Output to FILE instead of standard OUT ***************
       **          used with System.out.println("...");              **
      System.setOut (new PrintStream ( new FileOutputStream ("file.txt")));
      ****************************************************************/

      String s;
      s = file.readLine();

      while (!s.equals("#")) {

          String[][] circle = new String[5][13]; // board
          String current;  // current card
          int curval;      // int value of current card from getValue();
          int counter = 1; // number of moves (for problem output)

          //initialize first row which says which card is on top
          for (int i = 0; i < 13; i ++ ) {
            circle[0][i] = "1";
          }

          // read formatted input (4 rows of 13 cards in REVERSE order)
          for (int l = 0; l < 4; l ++) {
            StringTokenizer st = new StringTokenizer(s);
            for (int c = 0; c < 13; c ++) {
                circle [l+1][12-c] = st.nextToken();
            }
            s = file.readLine();
          }

          // initialize starting state
          current = circle [1][12];
          circle [0][12] = "2";
          curval = getValue(current) - 1;
          // loop according to rules of game
          while (getValue(circle[0][curval]) < 5) {
            counter++;
            current = circle[getValue(circle[0][curval])][curval];
            circle [0][curval] = ""+(getValue(circle[0][curval]) +1);
            curval = getValue(current) -1 ;
          }

          // problem output
          System.out.println(counter+","+current);
      }
    }

    // returns an integer value rep. by the first char of a string
    // effectively returns the value of a card -and- used in denoting
    // which card is on top
    public static int getValue (String s) {
      char ch = s.charAt(0);
      int i = 0;
      switch (ch) {
      case 'A': i = 1; break;
      case 'J': i = 11; break;
      case 'Q': i = 12; break;
      case 'K': i = 13; break;
      case 'T': i = 10; break;
      default: i = Integer.parseInt(""+s.charAt(0)); break;
      }

      return i;
    }

}
```

## Pipefitters – optimize geometric fit (Java)

```
/* Arel Cordero - Fall 2001 */

import java.io.*;
import java.util.*;

class pipefitters121 {

    public static void main (String[] args) throws Exception {
      BufferedReader file = new BufferedReader ( new FileReader ("121.dat") );
      String line = file.readLine();
      while (line != null) {
          // not put in while to avoid null pointer exeption
          if (line.equals("")) return;
          StringTokenizer st = new StringTokenizer( line );
          double width = Double.parseDouble(st.nextToken());
          double length = Double.parseDouble(st.nextToken());

          solve (width, length);

          line = file.readLine();
      }
    }

    public static void solve (double width, double length) {
      String way = "grid";
      int max = 0;

      //compute max by grid
      max = ((int) width) * ((int) length);

      for (int twice = 0; twice < 2; twice ++) {
          //width by length
          int w, l, mid, best;
          l = 1 + (int) ( (2*length - 2) / Math.sqrt(3) );
          w = (int) width;
          mid = (w - 1);

          if ((width-w) >= .5) mid ++;
          //NOTE the (double) type casting, l is int so division produces floor
          best = (int) ((Math.ceil( (double) l/2) * w) +
                     (Math.floor( (double) l/2) * mid));
          if (best > max) {
            way = "skew";
            max = best;
          }

          //repeat for length by width
          double t = width; width = length; length = t;
      }
      System.out.println(max+" "+way);
    }
}
```

## Radio Transmitters – optimal coverage (Java)

```
//Problem F, ACM NW regionals 2000
//Arel Cordero
import java.io.*;
import java.util.*;

class f {
    public static void main (String[] args) throws Exception {
       BufferedReader file = new BufferedReader( new FileReader ("f.dat"));
       String line = file.readLine();
       while (!line.equals("0 0 0")) {
           int[][] grid = new int[21][21];
           while (!line.equals("0 0 0")) {
              StringTokenizer st = new StringTokenizer (line);
              int x = Integer.parseInt(st.nextToken());
              int y = Integer.parseInt(st.nextToken());
              int p = Integer.parseInt(st.nextToken());

              //create starting scenario
              updateGrid (grid, x, y, p);
              line = file.readLine();
           }

           Vector v = getLackers(grid);
           solve(grid, v);
           line = file.readLine();
       }
    }

    // add the effect of one tower to the starting scenario
    public static void updateGrid (int[][] grid, int x, int y, int p) {
       //shift x and y !!! (because grid starts on -10)
       x += 10;     y += 10;
       for (int iy = 0; iy < 21; iy++) {
           for (int ix = 0; ix < 21; ix ++) {
              // double used instead of float to fix rounding error
              // delta (hypotenus) is left squrared instead
              //  of taking sqrt and later re-squaring
              double delta =  Math.pow( (x-ix), 2) + Math.pow( (y-iy), 2);
              int signal;
              if (delta == 0.0) {
                  signal = p;
              } else {
                  signal = (int) ( (double) p / delta);
              }
              grid[ix][iy] += signal;
           }
       }
    }

    //are there any weak reception areas?
    public static Vector getLackers (int[][] grid) {
       Vector lack = new Vector();

       for (int iy = 0; iy < 21; iy++) {
           for (int ix = 0; ix < 21; ix ++) {
              if (grid[ix][iy] < 100) {
                  int[] lacker = {ix, iy, grid[ix][iy]};
                  lack.add(lacker);
              }
           }
       }
       return lack;
    }
    public static void solve (int[][]grid, Vector v) {
       if (v.size() == 0) {
           System.out.println ("No additional transmitters needed");
           return;
       }

       int[] best = {0,0,1000000};

       for (int iy = 0; iy < 21; iy ++) {
           for (int ix = 0; ix < 21; ix ++) {
              int pow = lowest (ix, iy, v, best[2], grid);
              if (pow <= best[2]) {
                  if (pow == best[2]) {
                     //BREAK TIE
                     if ((ix < best[0])  || (ix == best [0] && iy < best[1])) {
                         best[0] = ix;
                         best[1] = iy;
                         best[2] = pow;
                     }
                  } else {
                     best[0] = ix;
                     best[1] = iy;
                     best[2] = pow;
                  }
              }
           }
       }

       System.out.println("Add a power "+best[2]+" transmitter at "+
                        (best[0] - 10)+ ","+(best[1]-10));
    }

    //determine the lowest power necessary at a given coordinate
    public static int lowest (int x, int y, Vector v, int bestL, int[][] grid) {
       int hi = 2000;
       int lo = 0;

       // find adequate upper bound
       while (!satisfiesV(x,y, hi, v, grid)) {
           hi *= 1.75;
       }
       //find hi
       while ( (hi-lo) > 1) {
           if (satisfiesV (x,y, (hi+lo)/2, v, grid) ) {
              hi = (hi+lo)/2;
           } else {
              lo = (hi+lo)/2;
           }
       }
       return hi;
    }

    //check if a power of p would satisfy the problem
    public static boolean satisfiesV (int x, int y, int p, Vector v, int[][] grid) {
       for (int i = 0 ; i < v.size(); i++) {
           double delta = Math.pow (   (x- ((int[])v.get(i))[0] ), 2) +
              Math.pow (   (y- ((int[])v.get(i))[1] ), 2);
           int signal;
           if (delta == 0) {
              signal = p+((int[]) v.get(i))[2];
           } else {
              signal = (int)  ( (double) p / delta) + ((int[]) v.get(i))[2];
           }
           if (signal < 100) return false;
       }
       return true;
    }
}
```

## Department of Redundancy department – inference table (C)

```
/* @JUDGE_ID: 14703TH 219 C */
/* Tim Singer - Fall 2001 */
#include "stdio.h"
#define getch() (getc(stdin))
/* input/derived (fd1/fd2) conditions -
   up to 100 FDs with up to 26 letters on each side.
   fd[n][0] = #letters in this FD */
char fd1[100][27];
char fd2[100][27];
int nfd;              /* # of FDs */
/* inference matrix:
   infer[i][100]=n (!=0) -> i can be inferred using n FDs numbered in infer[i][0..n-1] */
char infer[27][101];

/* copy FD #s from one row to another, not repeating */
void mergeinferrows(row1,row2) {
        int i,j,in;
        for(i=0;i<infer[row2][100];i++) {
                in=0;
                for(j=0;j<infer[row1][100];j++) {
                        if(infer[row1][j]==infer[row2][i]) in=1;
                }
                if(!in) {
                        infer[row1][j] = infer[row2][i];
                        infer[row1][100]++;
                }
        }
}

/* determine if FD n redundant by trying to come up with result of FD n w/o using FD n */
int solve(int n) {
    int i,j,k,changed,met;
        char dep,eff;
   /* blank out the infer matrix */
        for(i=0;i<27;i++) for(j=0;j<101;j++) infer[i][j]=0;
   /* copy the initial conditions from FD n to the infer matrix */
        for(i=0;fd1[n][i];i++) {
                infer[fd1[n][i]-1][0]=n;
                infer[fd1[n][i]-1][100]=1;
        }
    j=0; /* j = current FD */
   /* cycle through the FDs, updating the inferences until nothing is changed */
        do {
                changed=2;
                if(j!=n) {  /* do not use FD n */
                        met=1;
                    /* check for all input conditions */
                        for(i=0;fd1[j][i];i++) {
                                if(!(infer[fd1[j][i]-1][100])) met=0;
                        }
                        if(met) {
                           /* update effects */
                                for(i=0;fd2[j][i];i++) {
                                        eff=fd2[j][i]-1;
                                        if(!(infer[eff][100])) {
                                           /* Copy table values from dependencies met */
                                                for(k=0;fd1[j][k];k++) {
                                                        dep=fd1[j][k]-1;
                                                        mergeinferrows(eff,dep);
                                                }
                                                infer[eff][infer[eff][100]]=j; infer[eff][100]++;
                                                changed=1;
                                        }
                                }
                        }
```

```
             }
             j++;
             if(j==nfd) { j=0; if(changed==2) changed=0; else changed=2; }
    } while(changed);
   /* Check for all effects of FD(n) (and merge FD #s into one row) */
       met=1;
       for(i=0;fd2[n][i];i++) {
             if(!(infer[fd2[n][i]-1][100])) met=0;
             mergeinferrows(26,fd2[n][i]-1);
       }
   /* If all effects of FD(i) are covered: */
       if(met) {
      /* Say FD(i) is redundant and return */
       printf("    FD %i is redundant using FDs:",n+1);
         for(i=0;i<infer[26][100];i++) {
               if(infer[26][i]==n) continue;
          printf(" %i",infer[26][i]+1);
         }
         printf("\n");
         return 1;
       }
       return 0;
}

int main() {
       char c;
       int n,i,nsolved;
       int x;

       x=1;
       do {
         /* Read # of FDs */
            scanf("%i",&nfd);
            if(nfd==0) break;
         /* Read FDs */
            for(n=0;n<nfd; n++) {
                  c=getch();
                  i=0;
               /* read the input conditions */
                  while(c!='-') {
                        if(c>='A' && c<='Z') {
                              fd1[n][i]=c-'@'; i++; /* into numerical form */
                        }
                        c=getch();
                  }
                  fd1[n][i]=0;
                  i=0;
               /* read the derived values */
                  while(c!='\n') {
                        if(c>='A' && c<='Z') {
                              fd2[n][i]=c-'@'; i++;
                        }
                        c=getch();
                  }
                  fd2[n][i]=0;
            }
         /* For each FD(i): solve(i) */
            printf("Set number %i\n",x); x++;
            nsolved=0;
            for(n=0; n<nfd; n++) {
                  nsolved+=solve(n);
            }
            if(nsolved==0) printf("No redundant FDs.");
       printf("\n");
       } while(1); /*forever*/
       return 1;
}
```

## Node too Far – find unreachable nodes wrt path length (C)

```
/* James Marr - Fall 2001 */
/* Given a non-directed graph, finds number of nodes not reachable
   from a specific node with a given maximum path length
   (all paths have length 1) */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
/* @JUDGE_ID: 14709PJ 336 C */

struct node;
typedef struct node
{
  int visited;
  struct node *links[900];
  int numlinks;
}node;
node nodes[30];
int numnodes;

/* this is a stupid output requirement, ignore it */
int casecount = 1;

/* this is a simple hash table implementation
   the text file will reference nodes with arbitrary, integer names.
   calling lu with one of these names will return an index into nodes[] */

int hashname[30];
int numhash;
int lu(int name)
{
  int i;
  for(i=0; i<numhash; i++)
    {
      if(hashname[i] == name)
        return i;
    }
  hashname[i] = name;
  numhash++;
  return i;
}

/* recusively visits nodes, decreasing time to live (ttl) each time */
void visit(node *n, int ttl)
{
  n->visited = 1;
  if(ttl > 0)
    {
      int i;
      for(i=0; i<n->numlinks; i++)
        {
          visit(n->links[i], ttl-1);
        }
    }
}

int main(void)
{
  int i;
  char buf[256];
  char *s = buf;
  while(1)
    {
      int numlinks;
      fgets(buf, sizeof(buf), stdin);
      numlinks = strtol(buf, NULL, 10);
      if(!numlinks)
        break;
      numnodes = 0;
      numhash = 0;

      fgets(buf, sizeof(buf), stdin);
      s = buf;
      /* read in the links */
      for(i=0; i<numlinks; i++)
        {
          int name1, name2, id1, id2;
          char *old_s = s;
          errno = 0;
          name1 = strtol(s, &s, 10);

          /* messed up input format. the list of paths
             can be split up over multiple lines. this hacks
             around strtol to find out when we need a new line */
          if (*s == '\n' || ((*old_s) && (!*s))) {
            fgets (buf, sizeof (buf), stdin);
            s = buf;
            name1 = strtol (s, &s, 10);
          }

          name2 = strtol(s, &s, 10);
          id1 = lu(name1);
          id2 = lu(name2);
          if(id1 >= numnodes)
            {
              numnodes++;
              nodes[id1].numlinks = 0;
            }
          nodes[id1].links[nodes[id1].numlinks++] = &nodes[id2];
          if(id2 >= numnodes)
            {
              numnodes++;
              nodes[id2].numlinks = 0;
            }
          nodes[id2].links[nodes[id2].numlinks++] = &nodes[id1];
        }
      fgets(buf, sizeof(buf), stdin);
      s = buf;
      /* do each querry as to nodes reachable with a maximum path length */
      while(1)
        {
          int start, ttl, num;
          start = strtol(s, &s, 10);
          ttl = strtol(s, &s, 10);
          if(start == 0 && ttl == 0)
            break;
          for(i=0; i<numnodes; i++)
            nodes[i].visited = 0;
          visit(&nodes[lu(start)], ttl);
          num = 0;
          for(i=0; i<numnodes; i++)
            {
              if(!nodes[i].visited)
                num++;
            }
          printf("Case %i: %i nodes not reachable from node %i with TTL = %i.\n",
casecount++, num, start, ttl);
        }
    }
  return 0;
}
```

## Burgers – combinations and probability – dynamic prog. (Java)

```
/* @JUDGE_ID: 14705RZ 557 JAVA "dynamic programming" */
/* Carl Howells - Fall 2001 */


import java.io.*;
import java.text.*;

class Main
{
    static StreamTokenizer tok = new StreamTokenizer(new InputStreamReader(System.in));

    static int readInt() throws Exception
    {
        tok.nextToken();
        return (int)tok.nval;
    }

    public static void main(String [] args) throws Exception
    {
        double [][] prob = new double[501][501];

        // build a table of the results:
        for (int i = 0; i < 501; i++)
        {
            for (int j = 0; j < 501; j++)
            {
                if (i < 2 && j < 2)
                {
                    // if there aren't two of either left, there is no way
                    // to get two of the same as the last two
                    prob[i][j] = 0.0;
                }
                else if (i == 0 || j == 0)
                {
                    // if there are none of one left, but at least two of the
                    // other (because of the else), there is a 100% chance of
                    // the last two being the same
                    prob[i][j] = 1.0;
                }
                else
                {
                    // There are more than two of each.  Since the odds of
                    // the next burger being either type are the same, combine
                    // the two probibilities equally.
                    prob[i][j] = (prob[i - 1][j] + prob[i][j - 1]) / 2.0;
                }
            }
        }

        // create a DecimalFormat to match the output specification.
        DecimalFormat df = new DecimalFormat("0.0000");

        int times = readInt();
        for (int z = 0; z < times; z++)
        {
            // format and output the result at the correct location
            // in the result table
            int index = readInt() / 2;
            System.out.println(df.format(prob[index][index]));
        }
    } // main
} // class Main
```

## Burgers – combinations and probability – dynamic prog. (C)

```
/* @JUDGE_ID: 14705RZ 557 C "dynamic programming" */
/* Carl Howells - Fall 2001 */
#include <stdio.h>

int readInt()
{
    int result;
    scanf("%d", &result);
    return result;
}

double prob[501][501] = {0.0};

int main(int argc, char** argv)
{
    int i, j, times, z;

    /* build a table of the results: */
    for (i = 0; i < 501; i++)
    {
        for (j = 0; j < 501; j++)
        {
            if (i < 2 && j < 2)
            {
                /* if there aren't two of either left, there is no way */
                /* to get two of the same as the last two */
                prob[i][j] = 0.0;
            }
            else if (i == 0 || j == 0)
            {
                /* if there are none of one left, but at least two of the */
                /* other (because of the else), there is a 100% chance of */
                /* the last two being the same */
                prob[i][j] = 1.0;
            }
            else
            {
                /* There are more than two of each.  Since the odds of */
                /* the next burger being either type are the same, combine */
                /* the two probibilities equally. */
                prob[i][j] = (prob[i - 1][j] + prob[i][j - 1]) / 2.0;
            }
        }
    }

    times = readInt();
    for (z = 0; z < times; z++)
    {
        /* format and output the result at the correct location */
        /* in the result table */
        int index = readInt() / 2;
        printf("%1.4f\n", prob[index][index]);
    }
}
```

## FastFood – optimization of node placement (Java)

```
/*Chen Zebin, Fall 2001 */ /* Fastfood - 662 */
import java.util.*;
import java.io.*;
import java.math.*;

public class FastFood662 {
  public FastFood662() {
  }
  public static void main(String[] args) throws Exception {
      FastFood662 fastFood6621 = new FastFood662();
      BufferedReader d = new BufferedReader(new InputStreamReader(System.in));
      StringTokenizer st;
      String temp;
      Vector tv=new Vector();
      int []arrayn, arrayk;

      temp=d.readLine();
      int n, k;
      st=new StringTokenizer(temp);
      n=Integer.parseInt(st.nextToken());
      k=Integer.parseInt(st.nextToken());
      int counter=0;
      do {
          temp=d.readLine();
          if (temp==null) {
              break;
          }
          temp=temp.trim();
          if (temp.equals("0 0")) {
              break;
          }
          tv.addElement(temp);
          counter++;
      } while (true); // counter<n);
      arrayn=new int[n];
      for (int i=0; i<n; i++) {
          arrayn[i]=Integer.parseInt((String)tv.elementAt(i));
      }
      int loc=-1;
      int min=-1;
      int tempmin, temploc;
      for (int i=0; i<Math.pow(2, n); i++) {
          String ss=int2String(i, n);
          if (getTotalOccur(ss, '1')==k) {
              tempmin=getTotalDistance(ss, arrayn);
              temploc=i;
              if (min<0) {
                  min=tempmin; loc=temploc;
              }
              if (min>tempmin) {
                  min=tempmin; loc=temploc;
              }
          }
      }
      String sf=int2String(loc, n);
      int pp=0;
      for (int i=0; i<k; i++) {
          pp=sf.indexOf('1', pp);
          System.out.print("Depot "+(i+1)+" at restaurant "+(pp+1)+" serves restaurant");
          Vector vv=new Vector();
          for (int j=0; j<n; j++) {
              if (getSingleDepot(sf, j, arrayn)==pp) {
                  vv.addElement("" + (j+1));
              }
          }
          if (vv.size()>1) {
              System.out.println("s " + vv.elementAt(0) + " to " + vv.elementAt(vv.size()-
1));
          } else {
              System.out.println(" " + vv.elementAt(0));
          }
          pp++;
      }
  }
  public static int getSingleDepot(String s, int x, int[] arrayn) {
      int min=-1;
      int t;
      int loc=-1;
      for (int j=0; j<s.length(); j++) {
          if (s.charAt(j)=='1') {
              t=Math.abs(arrayn[j]-arrayn[x]);
              if (min<0) {
                  min=t; loc=j;
              }
              if (min>t) {
                  min=t; loc=j;
              }
          }
      }
      return loc;
  }
  public static int getTotalDistance(String s, int [] arrayn) {
      int counter=0;
      for (int i=0; i<arrayn.length; i++) {
          counter += getSingleDistance(s, i, arrayn);
      }
      return counter;
  }
  public static int getSingleDistance(String s, int x, int [] arrayn) {
      int min=-1;
      int t;
      for (int j=0; j<s.length(); j++) {
          if (s.charAt(j)=='1') {
              t=Math.abs(arrayn[j]-arrayn[x]);
              if (min<0) { min=t; }
              if (min>t) { min=t; }
          }
      }
      return min;
  }
  public static String int2String(int i, int n) {
      BigInteger bi=new BigInteger("" + i);
      String temp=bi.toString(2);
      for (int j=temp.length(); j<n; j++) {
          temp="0" + temp;
      }
      return temp;
  }
  public static int getTotalOccur(String s, char ch) {
      int counter=0;
      int fromIndex=0;
      do {
          int i=s.indexOf(ch, fromIndex);
          if (i>=0) {
              counter++; fromIndex=i+1;
          } else {
              break;
          }
      } while (true);
      return counter;
  }
}
```

## Shipping Routes – optimization of weighted paths (Java)

```
/**
 * Solution for the 5th week problem: 383 Shipping Routes
 * Author: Chen Zebin, zbchen@cs,
 */
import java.util.*;
import java.io.*;
import java.math.*;

public class Ship383 {
    public static boolean debug=false;
       public static void main(String args[]) throws Exception {
           if (args.length>0 && args[0].equals("-debug"))
              debug=true;
           Hashtable ht=new Hashtable();
           Vector v=new Vector(), tv1=new Vector(), tv2=new Vector(), tv3=new Vector();
           int M, N, P;
           BufferedReader d=new BufferedReader(new InputStreamReader(System.in));
           String temp;
           StringTokenizer st;

           int total;
           temp=d.readLine();
           total=Integer.parseInt(temp.trim());
           for (int totali=0; totali<total; totali++) {
              temp=d.readLine().trim();
              st=new StringTokenizer(temp);
              M=Integer.parseInt(st.nextToken());
              N=Integer.parseInt(st.nextToken());
              P=Integer.parseInt(st.nextToken());
              temp=d.readLine().trim();
              tv1=new Vector();
              st=new StringTokenizer(temp);
              while (st.hasMoreTokens()) {
                  tv1.addElement(st.nextToken());
              }
              tv2=new Vector();
              for (int i=0; i<N; i++) {
                  temp=d.readLine().trim();
                  tv2.addElement(temp);
              }
              tv3=new Vector();
              for (int i=0; i<P; i++) {
                  temp=d.readLine().trim();
                  tv3.addElement(temp);
              }
              v.addElement(tv1); v.addElement(tv2); v.addElement(tv3);
           }
           System.out.println("SHIPPING ROUTES OUTPUT");
           for (int totali=0; totali<total; totali++) {
              ht=new Hashtable();
              tv1=(Vector)v.elementAt(totali*3 );
              tv2=(Vector)v.elementAt(totali*3 + 1);
              tv3=(Vector)v.elementAt(totali*3 + 2);
              for (int i=0; i<tv1.size(); i++) {
                  temp=(String)tv1.elementAt(i);
                  MyNode bb=new MyNode(temp);
                  ht.put(temp, bb);
              }
              for (int i=0; i<tv2.size(); i++) {
                  temp=(String)tv2.elementAt(i);
                  st=new StringTokenizer(temp);
                  String a=st.nextToken();
                  String b=st.nextToken();
                  MyNode mn=(MyNode)ht.get(a);
                  mn.addNeighbour(b);
```

```
                ht.put(a, mn);
                mn=(MyNode)ht.get(b);
                mn.addNeighbour(a);
                ht.put(b, mn);
            }
            System.out.println("\nDATA SET " + (totali+1) + "\n");
            for (int i=0; i<tv3.size(); i++) {
                st=new StringTokenizer(((String)tv3.elementAt(i)).trim());
                int size=Integer.parseInt(st.nextToken());
                String a=st.nextToken();
                String b=st.nextToken();
                Properties prop=new Properties();
                prop.setProperty(a, a);
                int value=getShortest(a, b, ht, prop);
                if (value<0)
                   System.out.println("NO SHIPMENT POSSIBLE");
                else
                   System.out.println("$" + (100 * value * size));
            }
         }
          System.out.println("\nEND OF OUTPUT");
      }
      public static int getShortest(String a, String b, Hashtable ht, Properties prop) {
          //if (debug) System.out.println(a + " " + b + " is under processing.");
          Properties tprop=new Properties();
          Enumeration e=prop.keys();
          while (e.hasMoreElements()) {
             String key=(String)e.nextElement();
             tprop.setProperty(key, key);
          }
          MyNode aa=(MyNode)ht.get(a);
          if (aa.isNeighbour(b)) {
             return 1;
          } else {
             int min=-1000;
             Enumeration enum=aa.neighbours.keys();
             while (enum.hasMoreElements()) {
                 String temp=(String)enum.nextElement();
                 if (tprop.getProperty(temp)!=null) continue;
                 tprop.setProperty(temp, temp);
                 tprop.setProperty(a, a);
                 tprop.setProperty(b, b);
                 int tt=1+getShortest(temp, b, ht, tprop);
                 if (tt<0) continue;
                 if (min<0) min=tt;
                 if (min>tt) min=tt;
             }
             return min;
          }
      }
}

class MyNode {
    public Properties neighbours;
    String key;
    public MyNode(String key) {
       this.key=key;
       neighbours=new Properties();
    }
    public void addNeighbour(String b) {
       neighbours.setProperty(b, b);
    }
    public boolean isNeighbour(String b) {
       if (neighbours.getProperty(b)!=null) return true;
       return false;
    }
}
```

## Determine if directed graph is a tree   (Java)

```
/**
 * Solution for the 5th week problem: 615 Is It A Tree?
 * Author: Chen Zebin, zbchen@cs,
 */
import java.io.*;
import java.util.*;

public class Tree2 {
  public static boolean debug=false;

  public Tree2() {
  }
  public static void main(String[] args) throws Exception {
      if (args.length>0 && args[0].equals("-debug")) {
          debug=true;
      }
      Vector v=new Vector(), tv=new Vector();
      BufferedReader d = new BufferedReader(new InputStreamReader(System.in));
      String temp, a, b;
      StringTokenizer st;
      Vector output=new Vector();
      Hashtable ht=new Hashtable();

      /**
       * Read input data and save them to buffer for future processing -
       *   separate I/O and tree decision.
       */
      do {
          temp=d.readLine();
          if (temp==null) break;
          temp=temp.trim();
          if (temp.startsWith("-1")) { break; }
          if (temp.equals("")) { continue; }
          st=new StringTokenizer(temp);
          while (st.hasMoreTokens()) {
              a=st.nextToken();
              b=st.nextToken();
              if (a.equals("0") && b.equals("0")) {
                  v.addElement(tv);
                  tv=new Vector();
                  break;
              } else {
                  tv.addElement(a);
                  tv.addElement(b);
              }
          }
      } while (true);

      /**
       * A tree can be fully decided by the following condition:
       *   1. every node can go up to look for oldest ancestor; if all nodes have same
       *   oldest ancestors (root), is a tree. Otherwise is not a tree.
       *   2. every node has at most one father. Specially, for root, no father;
       *   for internal node as well as leaf, one father.
       */
      for (int i=0; i<v.size(); i++) {
          tv=(Vector)v.elementAt(i);
          boolean ifTree=true;
          ht=new Hashtable();
          for (int j=0; j<tv.size();) {
              a=(String)tv.elementAt(j);
              b=(String)tv.elementAt(j+1);
              j+=2;
              TreeNode tn=(TreeNode)ht.get(b);
              /**
```

```
                * if a node have more than one father, NOT a tree.
                */
               if (tn!=null) {
                   ifTree=false;
                   break;
               } else {
                   ht.put(b, new TreeNode(b, a));
               }
           }
           Enumeration enum=ht.keys();
           String key0=(String)enum.nextElement();
           String root0=getRootKey(ht, key0);
           while (enum.hasMoreElements()) {
               String key1=(String)enum.nextElement();
               String root1=getRootKey(ht, key1);
               /**
                * if a node have more than one root, NOT a tree.
                */
               if (!root0.equals(root1)) {
                   ifTree=false;
                   if (debug)
                       System.out.println("dupliacate root: " + key0 + " " + key1);
                   break;
               }
           }
           if (ifTree) {
               output.addElement(" ");
           } else {
               output.addElement(" not ");
           }
       }
       for (int i=0; i<output.size(); i++) {
           System.out.println("Case " + (i+1) + " is" + output.elementAt(i) + "a tree.");
       }
   }
   public static String getRootKey(Hashtable ht, String key) {
       if (debug)
           System.out.print("key=" + key + "; ");
       String value=key;
       Object tn=ht.get(value);
       while (tn!=null) {
           value=((TreeNode)tn).father;
           tn=ht.get(value);
       }
       if (debug)
           System.out.println("root is " + value);
       return value;
   }
}

class TreeNode {
  public String key, father;
  public Vector children;
  private String NAK="-1";
  public TreeNode(String key, String father) {
    this.key=key;
    this.father=father;
    this.children=new Vector();
  }
  public TreeNode(String key) {
    this.key=key;
    this.father=NAK;
    this.children=new Vector();
  }
}
```

## Longest Paths (Java)

```
/**
 * Solution for 6th week problem: 10000 Longest Paths
 * Author: Chen Zebin, zbchen@cs,
 *
 * Remind: the sample output seems error. For case 3: The longest path
 *  from 5 has length 3 (but not 2), e.g.,5->2->4->1
 */

import java.util.*;
import java.io.*;
import java.math.*;

public class Long10000 {
    public static void main(String args[]) throws Exception  {
      BufferedReader d=new BufferedReader(new InputStreamReader(System.in));
      String temp;
      Vector v=new Vector();
      Vector tv;
      Hashtable ht=new Hashtable();
      StringTokenizer st;

      do {
          tv=new Vector();
          temp=d.readLine().trim();
          if (temp.equals("0")) {
            break;
          }
          int totalNode=Integer.parseInt(temp);
          tv.addElement(temp);
          tv.addElement(d.readLine().trim());
          temp=d.readLine();
          while (!temp.equals("0 0")) {
            tv.addElement(temp);
            temp=d.readLine();
          }
          v.addElement(tv);
      } while (true);
      for (int ibig=0; ibig<v.size(); ibig++) {
            ht=new Hashtable();
          tv=(Vector)v.elementAt(ibig);
          int totalNode=Integer.parseInt((String)tv.elementAt(0));
          String start=(String)tv.elementAt(1);
          for (int i=2; i<tv.size(); i++) {
            temp=(String)tv.elementAt(i);
            st=new StringTokenizer(temp);
            String a=st.nextToken();
                String b=st.nextToken();
                MyNode mn=(MyNode)ht.get(a);
                if (mn==null)
                    mn=new MyNode(a);
                mn.addNeighbour(b);
                ht.put(a, mn);
                mn=(MyNode)ht.get(b);
                if (mn==null)
                    mn=new MyNode(b);
                mn.addNeighbour(a);
                ht.put(b, mn);
            }
            Properties prop=new Properties();
            Vector tempV=getLongest(start, prop, ht);
            System.out.println("The longest path from " + start + " has length "
+ tempV.elementAt(0) + ", finishing at " + tempV.elementAt(1));
        }
    }
    public static Vector getLongest(String key, Properties prop, Hashtable ht) {
            Properties tprop=new Properties();
      Enumeration enum=prop.keys();
      while (enum.hasMoreElements()) {
          String temp=(String)enum.nextElement();
          tprop.setProperty(temp, temp);
      }
      tprop.setProperty(key, key);
      MyNode mn=(MyNode)ht.get(key);
      enum=mn.neighbour.keys();

      int max=0;
      String last=key;
      while (enum.hasMoreElements()) {
          String temp=(String)enum.nextElement();
          if (tprop.getProperty(temp)!=null) {
            continue;
          }
          Vector v=getLongest(temp, tprop, ht);
          int tmax=1+Integer.parseInt((String)v.elementAt(0));
          if (max<tmax) {
            max=tmax;
            last=(String)v.elementAt(1);
          }
      }
      Vector v=new Vector();
      v.addElement("" + max);
      v.addElement(last);
      return v;
    }
}


class MyNode {
    public Properties neighbour;
    String key;
    public MyNode(String key) {
      this.key=key;
      neighbour=new Properties();
    }
    public void addNeighbour(String b) {
      neighbour.setProperty(b, b);
    }
    public boolean isNeighbour(String b) {
      if (neighbour.getProperty(b)!=null) {
          return true;
      }
        return false;
    }
}
```

## Quadratic Equation (mod p)—C++

```
/* Tim Singer - Fall 2001 */

/*****************************************************************************
PROBLEM:  Given quadratic equation (ax^2 + bx + c) and prime p,
   determine the roots (mod p), if they exist.
BONUS:  Implementation of a C++ big integer class
*****************************************************************************/

#include <iostream.h>
#include <stdio.h>
#include <string.h>

#ifndef ulong
#define ulong unsigned long
#endif

class bigint { //unsigned 128-bit number

public:

  bigint() { v[0]=v[1]=v[2]=v[3]=0; }  //=0
  bigint(ulong x) { v[0]=x; v[1]=v[2]=v[3]=0; } //=x
  bigint(const bigint& b) { set(b); } //=b
  bigint(char* c) { //read from c digit by digit
    set(0UL);
    for(int i=0;c[i]>='0' && c[i]<='9';i++) {
      mulmod(10);
      addmod((ulong)(c[i]-'0'));
    }
  }

  static void setp(bigint& b) { p = b; } //p=0 ==> no modulus

  void set(const bigint& b) {
    for(int i=0;i<4;i++) v[i]=b.v[i];
  }

  void add(const bigint& b) {
    int cy=0; ulong r=0;
    for (int i=0; i<4; i++) {
      r=v[i]+b.v[i]+cy; cy=0;          //add next digit
      if(r<v[i] || r<b.v[i]) cy=1;     //set carry if overflow
      v[i]=r;                          //writeback digit
    }
  }

  void sub(const bigint& b) {              // nearly the same as add
    int bw=0; ulong r=0;
    for (int i=0; i<4; i++) {
      r=v[i]-b.v[i]-bw; bw=0;
      if(r>v[i]) bw=1;
      v[i]=r;
    }
  }
```

```
void addmod(const bigint& a) {
  add(a); if(!p.zero()) while(!less(p)) sub(p);
}

void submod(const bigint& s) {
  sub(s); if(!p.zero()) while(!less(p)) add(p); //unsigned arithmetic
}

//Russian peasant multiplication (successive doubling)
void mulmod(const bigint& m) {
  bigint c(*this), d(m);
  set(0UL);
  while(!d.zero()) {
    if(d.v[0]&1) addmod(c);
    c+=c;
    d.shr();
  }
}

//Russian peasant exponentiation (successive squaring)
void powmod(const bigint& e) {
  bigint c(*this), d(e);
  set(1UL);
  while(!d.zero()) {
    if(d.v[0]&1) mulmod(c);
    c*=c;
    d.shr();
  }
}

void shr(void) {  //128-bit shift right
  int cy=0, cy2;
  for(int i=3; i>=0; i--) {
    cy2 = v[i]&1 << 31;
    v[i]>>=1; v[i]|=cy; cy=cy2;
  }
}

void inverse() { //multiplicitave inverse: x * x^(p-2)=1 (mod p)
  bigint pm2(p-bigint(2));
  powmod(pm2);
}

int less(const bigint& b) {  //this < b?
  for(int i=3;i>=0;i--) {
    if(v[i]<b.v[i]) return -1;
    if(v[i]>b.v[i]) return 0;
  }
  return 0;
}

int eq(const bigint& b) {  //this == b?
  for(int i=3;i>=0;i--) if(v[i]!=b.v[i]) return 0;
  return -1;
}

inline int zero() {  //this == 0?
  return v[0]==0 && v[1]==0 && v[2]==0 && v[3]==0;
}
```

```
int sqrt(void) {  //returns 0 if no sqrt
  bigint one(1);
  bigint pm1(p-one);
  bigint pm1d2(pm1); pm1d2.shr();
  bigint t(*this);

  //determine if it has square roots
  t.powmod(pm1d2);     // t = a^((p-1)/2)

  if(t!=one) return 0;

  //s = (p-1)>>XX, e=XX
  bigint s(pm1), e;
  while(!(s.v[0]&1)) { s.shr(); e+=one; }

  //a^=((s+1)/2), b=a^s
  bigint b(*this); b.powmod(s);
  bigint x(s); x+=one; x.shr(); powmod(x);
  if(b==one) return 1;

  //find n
  bigint n(2); t.set(0UL);
  for(;t!=pm1;n+=one) {
    t=n; t.powmod(pm1d2);
  }

  //g=n^s, r=e
  bigint g(n),r(e); g.powmod(s);
  bigint y;

  while(1) {
    y=b;
    bigint m(0UL);
    for(;m<r && y!=one;m+=one) {
      y*=y;
    }
    if(m.zero()) return 1;

    bigint z(r-m-one), h(g), i;

    for(;i<z;i+=one) h*=h;

    mulmod(h);
    g=h*h;
    b*=g;
    r=m;
  }
}


//operator definitions - not necessary, but helpful
// (and used extensively in this example)
inline bigint operator+(const bigint& a) {
  bigint b(*this); b.addmod(a); return b;
}

inline bigint& operator+=(const bigint& a) {
  addmod(a); return *this;
}
```

```
inline bigint operator-(const bigint& a) {
  bigint b(*this); b.submod(a); return b;
}

inline bigint& operator-=(const bigint& a) {
  submod(a); return *this;
}

inline bigint operator*(const bigint& a) {
  bigint b(*this); b.mulmod(a); return b;
}

inline bigint& operator*=(const bigint& a) {
  mulmod(a); return *this;
}

inline bigint operator/(const bigint& a) {
  bigint c(a); c.inverse();
  bigint b(*this); b.mulmod(c); return b;
}

inline bigint& operator/=(const bigint& a) {
  bigint c(a); c.inverse();
  mulmod(c); return *this;
}

inline bigint& operator=(const bigint& a) {
  set(a); return *this;
}

inline int operator<(const bigint& a)  { return less(a); }
inline int operator>=(const bigint& a) { return !less(a); }
inline int operator>(const bigint& a)  { return !(less(a) || eq(a)); }
inline int operator<=(const bigint& a) { return (less(a) || eq(a)); }
inline int operator==(const bigint& a) { return eq(a); }
inline int operator!=(const bigint& a) { return !eq(a); }

//reduction to 32-bit unsigned int
operator ulong(void) { return v[0]; }

//conversion to string
operator char*(void) {
  //remove modulus temporarily
  bigint ps(p); p.set(bigint(0UL));
  char* x=new char[40]; //39 digits max
  x[0]='0'; x[1]=0;  //default = "0"
  int i=0;
  if(!zero()) recur_set(x,i,bigint(1),bigint(*this));
  x[i]=0;  //terminate the string
  //restore modulus
  p=ps;
  return x;
}
```

```
protected:

  ulong v[4];
  static bigint p;

  //utility for conversion to char*
  void recur_set(char* c, int& i, bigint base, bigint& val) {
    if(val>=base) {
      if(i<38)
        recur_set(c,i,base*bigint(10),val);
      c[i]='0';
      while(base<=val) { val-=base; c[i]++; }
      i++;
    }
  }

};    // class bigint
```

```
/**************************** MAIN PROBLEM *********************************/

bigint bigint::p(0UL);  //init p to 0

int val(char* s) {
  if(strlen(s)>39) return 0;
  for(unsigned i=0; i<strlen(s); i++)
    if(s[i]<'0' || s[i]>'9') return 0;
  return -1;

}

int main() {
  char s[4][256];
  scanf("%s %s %s %s",s[0],s[1],s[2],s[3]);
  while(!feof(stdin)) {
    printf("Q[x_] := Mod[ %sx^2 + %sx + %s, %s ]\n{ ",s[0],s[1],s[2],s[3]);

    if(!val(s[0]) || !val(s[1]) || !val(s[2]) || !val(s[3])) {
      printf("invalid input");
    } else {
      bigint::setp(bigint(0UL));
      bigint a(s[0]), b(s[1]), c(s[2]), p(s[3]);
      bigint::setp(p);

      //x = (-b + sqrt(b^2 - 4ac))/2a
      //compute a sqrt - might be nonexistant
      //note: the other square root is symmetric
      //    i.e. sqrt1(x) = p - sqrt2(x)

      bigint sq(b*b), t(a*c*bigint(4));
      sq-=t;
      if(sq.zero() || sq.sqrt()) {
        t=bigint(2)*a;
        if(!t.zero()) {
          bigint a1, a2;
          a1-=b; a2-=b;
          a1+=sq; a2-=sq;
          a1/=t; a2/=t;
          char *a1s=(char*)a1, *a2s=(char*)a2;
          if(a1==a2) {
            printf("%s",a1s);
          } else {
            printf("%s, %s",a1s,a2s);
          }

          delete[] a1s;   delete[] a2s;
        } else {  //division by zero
          printf("has no roots");
        }
      } else {  //b^2+4ac has no square roots
        printf("has no roots");
      }
    }

    printf(" }\n\n");
    scanf("%s %s %s %s",s[0],s[1],s[2],s[3]);
  }
  return 0;
}
```

## Anagram – generate words from letters (C++)

```
/* Erin Keenan - Fall 2001 */
/* Anagram - problem 195 */
#include <stdio.h>
#include <stdlib.h>

//print out anagrams of given string alphebetically
//recursive dynamic-programming algorithm that sorts innately.

unsigned char whole_str[512];
void ana(unsigned char *str); //recursive anagram function
int main(void);

int main(void)
{
      int num;
      scanf("%d\n", &num);
      for(int i = 0; i < num; ++i)
    {
            gets((char*)whole_str);
            ana(whole_str);
    }
}

void ana(unsigned char *str)
{
      if(*(str + 1) == 0) //1 character string, just print
    {
            printf("%s\n",whole_str);
            return;
    }
      // keep finding lowest character alphabetically
      // and swap it to the front, then call ourselves
      // (this is the 'clever' dynamic programming part --
      //  we don't consider all variations of position,
      //  just alphebetically-unique variations)

      unsigned char oldCur = 0, cur = -1, *c, *curPtr;
      while(1)
    {
            c = str;
            cur = -1;
            while(*c) //find smallest character after oldCur
            {
                  if(*c > oldCur && *c < cur)
                  {
                        cur = *c;
                        curPtr = c;
                  }
                  ++c;
            }
            if((char)cur == -1) //used up all characters, return
                  return;
            //swap cur to front
            *curPtr = *str;
            *str = cur;
            oldCur = cur;
            //print out all possibilities
            ana(str + 1);
    }
}
```

## Expanding Fractions – decimal reps of rationals (C++)

```
/* Erin Keenan - Fall 2001 */
/* @JUDGE_ID: 14703TH 275 C++ */
/* print the decimal expansion of a fraction - tricky part is identifying if decimal
   repeats (and if so, what repeats) or if it terminates.
   use gradeschool division algorithm:
                                   _________
                       denominator|numerator

   see how many times denominator goes into numerator and record it as
   next digit in decimal expansion. then subtract digit*denominator from
   numerator, save result as new numerator. rinse, repeat.

   * we locate repeating digits by keeping a list of all the values of numerator.
   each time we calculate a new numerator, we check to see if we've encountered
   it before when adding to list. if we have, the decimal will repeat.
   * in this code, denominator is called 'denom' and the numerator 'ator'(sorry).
   * the online judge accepts this code, but says there is an output formatting
    error.  i've never been able to find it. . .
*/

#include <stdlib.h>
#include <stdio.h>

typedef struct ator_list
{
       int pos, ator; //pos = digit corrosponding to this ator
       ator_list *next;
};
ator_list *first_ator = 0, *repeat_ator = 0;
int curPos;
void output(char c);
bool insert_ator(int ator);
void kill_ators(void);
void doDiv(int ator, int den);
void div(int ator, int den, int *out_times, int *out_remain);

int main(void)
{
       while(1)
    {
              int ator, den;
              scanf("%i %i\n", &ator, &den);
              if(ator == 0)
                     return 0;
              //printf("ator: %i den: %i\n\n", ator, den);
              doDiv(ator, den);
              output(0);
              kill_ators();
              printf("\n");
    }
}

void doDiv(int ator, int den)
{
       curPos = 1;
       output('.'); //den is always > ator
       ator *= 10;
       int times, remain;
       while(1)
    {
              fflush(stdout);
              if(insert_ator(ator)) //repeat -- we have an infinite loop
              {
                     printf("\nThe last %i digits repeat forever.\n",
                            curPos-repeat_ator->pos);
```

```
                return;
            }
            //      printf("\nator: %i den: %i\n", ator, den);
            div(ator, den, &times, &remain);
            output(times + '0');
            //      printf("\ntimes: %i\n", times);
            if(!remain) //exact fraction, done
            {
                printf("\nThis expansion terminates.\n");
                return;
            }
        //printf("\ntimes %i remain %i ator %i den %i",times,remain,ator,den);
            if(times)
                ator = (ator - times*den)*10;
            else
                ator *= 10;
            //printf("\nator: %i\n", ator);
            ++curPos;
    }
}

bool insert_ator(int ator)
{
    //make sure not repeat
    ator_list *cur = first_ator;
    while(cur)
    {
            if(cur->ator == ator) //repeat
            {
                //  printf("\nrepeat ator %i\n", ator);
                repeat_ator = cur;
                return true;
            }
            cur = cur->next;
    }
    cur = new ator_list;
    cur->pos = curPos;
    cur->ator = ator;
    cur->next = first_ator;
    first_ator = cur;
    return false;
}

void kill_ators(void)
{
    first_ator = 0; //to do: not leak
}

void div(int ator, int den, int *out_times, int *out_remain)
{
    *out_times = ator/den;
    *out_remain = ator%den;
}

void output(char c)
{
    static int count = 0;
    if(c == 0)
    {
            count = 0;
            return;
    }
    if(count++ == 50)
    {
            count = 1;
            printf("\n");
    }
    printf("%c", c);
}
```

## Embedded Codes – find substrings (C++)

```
/* Erin Keenan - Fall 2001 */
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

//silly program to find a string embedded within a string like
//c9a9n9d9y (that yummy stuff your evil mother wouldn't let you eat)
//is hidden in between all those 9's. note that embedded string must be seperated
//by constant amount of garbade space. for example:
//c911a420n411d187y is legal, but c999a9n9d83y isn't.

//output: either the string is not found, or the number of seperating spaces.

int search(char *find, char *str);

int main(void)
{
      char str[90], find[90], buffer[90];
      while(1)
    {
            gets(find);
            if(!strcmp("#", find)) //exit
                  return 0;
            gets(str);

            find[strlen(find) - 1] = 0;
            str[strlen(str) - 1] = 0;

            int res = search(find, str);
            if(res == 0)
                  printf("[%s] is not found.\n", find);
            else
                  printf("[%s] is found with encoding of %i\n", find, res);
    }
      return 1;
}

int search(char *find, char *str) //returns 0 if not found, else pattern distance
{
      int strleng = strlen(str), findlen = strlen(find);

      //search for possible string start
      int i = 0;
      while(str[i])
    {
            if(str[i] == find[0])
            {
                  //possible match, search for next character
                  int k = i + 1, savek;
              try_again:
                  while(str[k])
                  {
                        if(str[k] == find[1]) //2nd character matches
                        {
                              break;
                        }
                        ++k;
                  }
                  savek = k;
                  if(str[k]) //have 2nd char match, check if whole pattern works
                  {
                        int offset = k - i;
                        int x = 2; //3rd character of find
                        while(x != findlen)
                        {
                              k += offset;
                              if(k >= strleng) //went off edge -- no match
                              {
                                    break;   //do we need to try again? no. . .
                              }
                              if(str[k] != find[x]) //pat broken-try new offset
                              {
                                    k = savek + 1;
                                    goto try_again;
                              }
                              ++x;
                        }
                        if(x == findlen) //whole patter matched, found string
                              return offset;
                        assert(k >= strleng); //shouldn't be here unless no match
                  }
            }
            ++i;
    }
      //not found, return 0
      return 0;
}
```

## Matty’s Blocks – perspective equivalence (C)

```
#include <stdio.h>
#include <stdlib.h>

/* @JUDGE_ID: 14709PJ 434 C */
/* Dan Stutzbach - Fall 2001 */
/* In this problem, you are given the size of square board, of max
   size 8x8.  Blocks are stacked on the board, and you are given the
   view from the front and from the right.  The view is a perfectly
   level one, such that the viewer can only perceive that largest
   number of blocks in a row. */

unsigned store[64]; /* Store the board */
unsigned f[8]; /* Front view */
unsigned r[8]; /* Right view */
unsigned rused[8]; /* explained later */
#define min(x,y) (((x) < (y)) ? (x) : (y))

void init_store (void) {
        memset (rused, 0, sizeof (rused));
        memset (store, 0, sizeof (store));
}

/* When you don't want to deal with C's syntax for multidimensional arrays.. */
void put (unsigned i, unsigned j, unsigned s) { store[i*8+j] = s; }
unsigned get (unsigned i, unsigned j) { return store[i*8+j]; }

/* Add up how many blocks are on the board */
unsigned total (void) {
        unsigned total = 0;
        unsigned i;
        for (i=0; i<64; i++) {
                total += store[i];
        }
        return total;
}

int main (void) {
        unsigned num_cases;
        char line[256];
        fgets (line, sizeof line, stdin);
        num_cases = strtoul (line, NULL, 10);
        while(num_cases--) {
                unsigned i, j, k, n, m;
                char *s;

                init_store ();
                /* Read in the view */
                fgets (line, sizeof line, stdin);
                k = strtoul (line, NULL, 10);
                fgets (line, sizeof line, stdin);
                s = line;
                for (i = 0; i < k; i++)
                        f[i] = strtoul (s, &s, 10);
                fgets (line, sizeof line, stdin);
                s = line;
                for (i = 0; i < k; i++)
                        r[i] = strtoul (s, &s, 10);
                /* We need to compute the minimal set of blocks that
                   can compose this view.  We start with each row in
                   the front view.  Within each row, we make several
                   passes. */
                for (i = 0; i < k; i++) {
                        /* First, look for a matching front and right
                           view, where we haven't use that row from
                           the right view yet.  */
                        for (j = 0; j < k; j++) {
                                if ((f[i] == r[j]) && !rused[j]) {
                                        put (i, j, f[i]);
                                        rused[j] = 1;
                                        goto placed;
                                }
                        }
                        /* If we've used all the matching right view
                           rows, then it doesn't matter which one we
                           pick.  Pick any matching right view.  */
                        for (j = 0; j < k; j++) {
                                if (f[i] == r[j]) {
                                        put (i, j, f[i]);
                                        goto placed;
                                }
                        }
                        /* If there aren't any matching right views,
                           just pick any right view that's larger than
                           the front view.  */
                        for (j = 0; j < k; j++) {
                                if (f[i] <= r[j]) {
                                        put (i, j, f[i]);
                                        goto placed;
                                }
                        }
                placed:;
                }
                /* Now, check all the right views, one row at a time */
                for (i = 0; i < k; i++) {
                        /* If this is set, we've already taken care of
                           this row */
                        if (rused[i]) continue;

                        /* If there's a front row with the same
                           height, use that */
                        for (j = 0; j < k; j++) {
                                if (f[j] == r[i]) {
                                        put (j, i, r[i]);
                                        goto placed2;
                                }
                        }
                        /* Otherwise, use any front row that's larger */
                        for (j = 0; j < k; j++) {
                                if (r[i] <= f[j]) {
                                        put (j, i, r[i]);
                                        goto placed2;
                                }
                        }
                placed2:;
                }
                /* We've go the minimum, store it in n */
                n = total ();
                /* The maximum case is actually much easier.  For each
                   coordinate, just use the smaller of the front and
                   rear view.  */
                for (i = 0; i < k; i++) {
                        for (j = 0; j < k; j++) {
                                put (i, j, min (f[i], r[j]));
                        }
                }
                m = total () - n;
                printf ("Matty needs at least %d blocks, and can add at "
                        "most %d extra blocks.\n", n, m);
        }
        return 0;
}
```

## Matrix class (C++)

```
/* Canned Matrix class with matrix multiplication...
   Does not bounds check insertion or retrieval.
*/
/* Dan Ellsworth - Fall 2001 */
class Matrix {
public:
   int* mat;
   int row;
   int col;

   Matrix(int row,int col);
   Matrix* mult(Matrix &a,Matrix &b);
   int get(int c,int r);
   void set(int c,int r,int v);
private:
};

Matrix::Matrix(int r,int c) {
   mat=new int[r*c];
   row=r;
   col=c;
   for(int i=0;i<r*c;i++) {
      mat[i]=0;
   }
}
Matrix* mult(Matrix *a,Matrix *b) {
   Matrix* ret=new Matrix(a->row,b->col);
   for(int i=0;i<a->row;i++) {
      for(int j=0;j<b->col;j++) {
         int tmp=0;
         for(int k=0;k<a->col;k++) {
            tmp=tmp+a->get(i,k)*b->get(k,j);
         }
         ret->set(i,j,tmp);
      }
   }
}

int Matrix::get(int r,int c) { return mat[r*col+c]; }
void Matrix::set(int r,int c,int v) { mat[r*col+c]=v; }

// Sample usage:
#include <iostream.h>
void main() {
   Matrix* ident=new Matrix(2,2);
   ident->set(0,0,1);
   ident->set(1,1,1);
   cout << ident->get(0,0) << ' ' << ident->get(0,1) << endl;
   cout << ident->get(1,0) << ' ' << ident->get(1,1) << endl << "------" << endl;
   Matrix* a=new Matrix(2,2);
   a->set(0,0,4);
   a->set(1,0,5);
   cout << a->get(0,0) << ' ' << a->get(0,1) << endl;
   cout << a->get(1,0) << ' ' << a->get(1,1) << endl << "------" << endl;
   Matrix* b=mult(a,ident);
   cout << b->get(0,0) << ' ' << b->get(0,1) << endl;
   cout << b->get(1,0) << ' ' << b->get(1,1) << endl << "------" << endl;
}
```

## Matrix class (Java)

```
/* Dan Ellsworth
Shrinkwrapped Matrix with multiplication
*/

class Matrix {
   public int[][] mat;
   public int row;
   public int col;

   public Matrix(int r,int c) {
      row=r;
      col=c;
      mat=new int[row][col];
   }

   public Matrix mult(Matrix b) {
      Matrix ret=new Matrix(row,b.col);
      for(int i=0;i<row;i++) {
         for(int j=0;j<b.col;j++) {
            int val=0;
            for(int k=0;k<col;k++) {
               val=val+mat[i][k]*b.mat[k][j];
            }
            ret.mat[i][j]=val;
         }
      }
      return ret;
   }

   public String toString() {
      String ret="";
      for(int i=0;i<row;i++) {
         for(int j=0;j<col;j++) {
            ret=ret+mat[i][j]+"\t";
         }
         ret=ret+"\n";
      }
      return ret;
   }

}
```

## Longest path in directed graph (C)

```
/* finds the LONGEST path in a directed, non-cyclical graph
   note that all edges have weight 1 */

/* @JUDGE_ID: 14709PJ 10000 C "blah" */
/* James Marr - Fall 2001 */

#include <stdio.h>

/* funky output requirement */
int curCase = 1;

struct node;
typedef struct node
{
  int key;
  int longest;
  struct node *links[1000];
  int numlinks;
}node;

node nodes[1000];
int numnodes;

/* lookup function. if a node with key is not found, one is created and returned
*/
node* lu(int key)
{
  int i;
  for(i=0; i<numnodes; i++)
    {
      if(nodes[i].key == key)
      return &nodes[i];
    }
  nodes[i].key = key;
  nodes[i].longest = 0;
  nodes[i].numlinks = 0;
  numnodes++;
  return &nodes[i];
}

/* visits a node given a length to this node
   if the path currently being traversed is longer than the path already
containing node n,
   the new path is used */
void visit(node *n, int length)
{
  if(length >= n->longest)
    {
      int i;
      n->longest = length;
      for(i=0; i<n->numlinks; i++)
      visit(n->links[i], length + 1);
    }
}

int main()
{
  while(1)
    {
      int pos, start, i;
      node *longest;
      scanf("%i", &pos); /* this field is worthless! */
      if(pos == 0) break;
      scanf("%i", &start);
      numnodes = 0;
      while(1)
      {
        int l1, l2;
        node *n;
        scanf("%i %i", &l1, &l2);
        if(l2 == 0 && l1 == 0) break;
        n = lu(l1);
        n->links[n->numlinks] = lu(l2);
        n->numlinks++;
      }
      visit(lu(start), 0);
      longest = &nodes[0];
      /* finds the longest path of all found paths */
      for(i=1; i<numnodes; i++)
      {
        if(nodes[i].longest > longest->longest ||
           (nodes[i].longest == longest->longest &&
            nodes[i].key < longest->key))
          longest = &nodes[i];
      }
      printf("Case %i: The longest path from %i has length %i, finishing at
%i.\n\n", curCase, start, longest->longest, longest->key);
      curCase++;
    }
}
```

## Correct Move – determine simple king/queen moves (C)

```
/* determinds if chess moves involving a rook and a king are valid
   translates between a row major, single indexed array representing a
multideminsional array using normal and modular devision
   imput consists of a king and queen (really a rook) position on an 8x8 board,
and a possible move for the queen */
/* James Marr - Fall 2001 */

#include <stdio.h>
#include <stdlib.h>

/* @JUDGE_ID: 14709PJ 255 C */

/* macros for converting between a single index 8x8 array and x,y coords */
#define x(a) ((a)%8)
#define y(a) ((a)/8)

/* imbetween */
#define in(x) ((x <= 7) && (x >= 0))

int legal (int kx, int ky, int qx, int qy)
{
        if (!in (kx) || !in (ky))
                return 0;
        if ((kx == qx) || (ky == qy))
                return 0;
        return 1;
}

int main (void)
{
        char line[256];

        while (fgets (line, sizeof line, stdin)) {
                unsigned ks, qs, qm;
                char *s = line;

            /* king start, queen start and queen move */
                ks = strtoul (s, &s, 10);
                qs = strtoul (s, &s, 10);
                qm = strtoul (s, &s, 10);

                if (ks == qs) {
                        printf ("Illegal state\n");
                        continue;
                }

                if ((x(ks) == x(qs)) && (x(qm) == x(ks))) {
                        if (y(qs) > y(qm)) {
                                if ((y(ks) >= y(qm)) && (y(qs) >= y(ks))) {
                                        printf ("Illegal move\n");
                                        continue;
                                }
                        } else if ((y(ks) >= y(qs)) && (y(qm) >= y(ks))) {
                                printf ("Illegal move\n");
                                continue;
                        }
                }

                if ((y(ks) == y(qs)) && (y(qm) == y(ks))) {
```

```
                if (x(qs) > x(qm)) {
                        if ((x(ks) >= x(qm)) && (x(qs) >= x(ks))) {
                                printf ("Illegal move\n");
                                continue;
                        }
                } else if ((x(ks) >= x(qs)) && (x(qm) >= x(ks))) {
                        printf ("Illegal move\n");
                        continue;
                }
        }

        if ((x(qm) != x(qs)) && (y(qm) != y(qs))) {
                printf ("Illegal move\n");
                continue;
        }

        if (qm == qs) {
                printf ("Illegal move\n");
                continue;
        }

        if (x(ks) == x(qm)) {
                if (abs (y(ks) - y(qm)) <= 1) {
                        printf ("Move not allowed\n");
                        continue;
                }
        }

        if (y(ks) == y(qm)) {
                if (abs (x(ks) - x(qm)) <= 1) {
                        printf ("Move not allowed\n");
                        continue;
                }
        }

        if ((!legal (x(ks) - 1, y(ks), x(qm), y(qm)))
            && (!legal (x(ks) + 1, y(ks), x(qm), y(qm)))
            && (!legal (x(ks), y(ks) - 1, x(qm), y(qm)))
            && (!legal (x(ks), y(ks) + 1, x(qm), y(qm)))) {
                printf ("Stop\n");
                continue;
        }

        printf ("Continue\n");
    }

    return 0;
}
```

## Graph connectivity – find connected subgraphs (C)

```
/* finds the number of connected sub-graphs in non-directed graphs */
/* @JUDGE_ID: 14709PJ 459 C */
/* James Marr - Fall 2001 */
#include <stdio.h>

struct node;
typedef struct node {
  char key; /* nodes are represented by a char */
  int visited;
  struct node *links[1000];
  int numlinks;
} node;
node nodes[26];
int numnodes = 0;

/* lookup function. if no node is found for key, a new node is created */
node *lu(char key) {
  int i;
  for(i=0; i<numnodes; i++)
      if(nodes[i].key == key) return &nodes[i];
  nodes[i].key = key;
  nodes[i].visited = 0;
  nodes[i].numlinks = 0;
  numnodes++;
  return &nodes[i];
}

/* recusively visits nodes, marking them as visited */
void visit(node *n) {
  if(!n->visited) {
      int i;
      n->visited = 1;
      for(i=0; i<n->numlinks; i++)
          visit(n->links[i]);
    }
}

int main() {
  int i, graphs;
  char line[4];
  fgets(line, sizeof line, stdin);
  for(i='A'; i<=line[0]; i++)
    lu(i);
  while(1) {
      node *n1, *n2;
      fgets(line, sizeof line, stdin);
      if(line[0] == '\n') break;
      n1 = lu(line[0]); n2 = lu(line[1]);
      n1->links[n1->numlinks] = n2;
      n1->numlinks++;
      n2->links[n2->numlinks] = n1;
      n2->numlinks++;
    }
  graphs = 0;
  for(i=0; i<numnodes; i++){
      if(!nodes[i].visited) {
        graphs++;
        visit(&nodes[i]);
       }
    }
  printf("%i\n", graphs);
  return 0;
}
```

## Moth Eradication – convex hull (Java)

```
/* Carl Howells - Fall 2001 */
/* Moth Eradication - problem 218 */
import java.io.*;
import java.util.*;
import java.text.*;

class Main
{
	static StreamTokenizer tok = new StreamTokenizer(new InputStreamReader(System.in));

	static int readInt() throws Exception	{
		tok.nextToken();
		return (int)tok.nval;
	}

	static double readDouble() throws Exception	{
		tok.nextToken();
		return tok.nval;
	}

	public static void main(String [] args) throws Exception {
		int region = 1;
		while (true) {
			int size = readInt();
			if (size == 0) break;

			List points = new ArrayList();

			double minX = Double.MAX_VALUE;
			double minY = Double.MAX_VALUE;

			for (int i = 0; i < size; i++) {
				double x = readDouble();
				double y = readDouble();

				points.add(new Point(x, y));

				// not actually finding the lowest x, but
				// the x that goes with the lowest y.
				if (y < minY) {
					minY = y;
					minX = x;
				}
			}

			// translate so that the lowest point is the origin
			Iterator it = points.iterator();
			while (it.hasNext()) {
				Point p = (Point)it.next();
				p.translate(-minX, -minY);
			}

			// Graham-Scan
			// This is the algorithm which actually finds convex hulls:

			// sort into clockwise ordering
			// check my point class to see how ordering is determined.
			Collections.sort(points);

			LinkedList stack = new LinkedList();

			// keep the index of the next point
			int i = 0;

			// scan through the points...  If the last three in
// the stack form a left turn, push another onto the
// stack, or terminate.  If they don't, pop the second
// to last and check again.
while (true) {
       if (stack.size() < 3) {
              stack.add(points.get(i++));
       }
       else {
              Point p3 = (Point)stack.removeLast();
              Point p2 = (Point)stack.removeLast();
              Point p1 = (Point)stack.removeLast();

              if (leftTurn(p1, p2, p3)) {
                     stack.add(p1);
                     stack.add(p2);
                     stack.add(p3);
                     if (i < points.size()) {
                            stack.add(points.get(i++));
                     }
                     else {
                            break;
                     }
              }
              else {
                     stack.add(p1);
                     stack.add(p3);
              }
       }
} // while

// check for degenerate case of colinear (with the origin)
// points at the end of the sorted list:

// point to check to see if others are colinear
// with this and origin
Point last = (Point)stack.getLast();

// move i to the second to last..
i -= 2;
while (i >= 0 && last.colinear((Point)points.get(i))) {
       stack.add(points.get(i));
       i--;
}

// translate points back to original location
it = stack.iterator();
while (it.hasNext()) {
       Point p = (Point)it.next();
       p.translate(minX, minY);
}

// do output
System.out.println("Region #" + region + ":");

double dist = 0.0;
Point prev = (Point)stack.getFirst();

ListIterator lit = stack.listIterator(stack.size());
while (lit.hasPrevious()) {
       Point curr = (Point)lit.previous();
       dist += prev.distance(curr);
       System.out.print(prev + "-");
       prev = curr;
}

System.out.println(prev);
DecimalFormat df = new DecimalFormat("###################.00");
System.out.println("Perimeter length = " + df.format(dist));
System.out.println();
```

```
                region++;
            } // while
        } // main

        static boolean leftTurn(Point p1, Point p2, Point p3){
            Point t1 = p2.minus(p1);
            Point t2 = p3.minus(p1);
            return t1.compareTo(t2) < 0;
        }

        static class Point implements Comparable {
            double x, y;

            static DecimalFormat df = new DecimalFormat("0.0");

            Point(double x, double y) {
                this.x = x;
                this.y = y;
            }

            public int compareTo(Object obj) {
                Point o = (Point)obj;

                // the origin should always be first
                if (x == 0.0 && y == 0.0) return -1;
                if (o.x == 0.0 && o.y == 0.0) return 1;

                // primary comparison is angle relative to origin
                double crossed = (x * o.y) - (o.x * y);

                if (crossed < 0.0) return 1;
                if (crossed > 0.0) return -1;

                // use distance as secondary comparison, with the closest
                // to the origin coming last.

                double d = x * x + y * y;
                double od = o.x * o.x + o.y * o.y;

                if (d < od) return -1;
                if (d > od) return 1;

                return 0;
            }

            void translate(double dx, double dy) {
                x += dx;
                y += dy;
            }

            Point minus(Point p) {
                return new Point(x - p.x, y - p.y);
            }

            public String toString() {
                return "(" + df.format(x) + "," + df.format(y) + ")";
            }

            double distance(Point p) {
                return Math.sqrt((x - p.x) * (x - p.x) + (y - p.y) * (y - p.y));
            }

            boolean colinear(Point p) {
                // colinear with respect to origin:
                return 0.0 == (x * p.y) - (p.x * y);
            }
        }
} // class Main
```

## Say Cheese – minimal path – Dijkstra (Java)

```
/* Carl Howells - Fall 2001 */
import java.io.*;
import java.util.*;

// This is a solution to Problem B from the 2001 ACM world finals.
// A description of the problem may be obtained from:
// http://icpc.baylor.edu/Past/icpc2001/Finals/problems.pdf

public class B
{
      // standard IO stuff
      private static StreamTokenizer stok;

      public static int nextInt() throws Exception {
            stok.nextToken();
            return (int)stok.nval;
      } // nextInt

      public static void main(String [] args) throws Exception {
            stok = new StreamTokenizer(new BufferedReader(new
FileReader("cheese.in")));

            int cheeseNumber = 1;
            while (true) {
                  // parse input for this data set
                  int numholes = nextInt();
                  if (numholes == -1) break;

                  Hole [] holes = new Hole[numholes + 1];

                  for (int i = 0; i < numholes; i++) {
                      holes[i]=new Hole(nextInt(),nextInt(),nextInt(),nextInt());
                  } // for

                  Hole finish = new Hole(nextInt(), nextInt(), nextInt());

                  Hole start = new Hole(nextInt(), nextInt(), nextInt());

                  // for simplification of algorithm, include the destination
                  // in the array
                  holes[numholes] = finish;


                  // set initial distances directly to start position
                  for (int i = 0; i < holes.length; i++)
                        holes[i].dist = distBetween(holes[i], start);


                  // dijkstra's algorithm
                  for (int i = 0; i < holes.length; i++) {
                        // find shortest distance not in use
                        int min = i;
                        for (int j = i + 1; j < holes.length; j++)
                              if (holes[j].dist < holes[min].dist) min = j;

                        // swap to first unused position
                        Hole temp = holes[min];
                        holes[min] = holes[i];
                        holes[i] = temp;
                        // relax with new point
                        for (int j = i + 1; j < holes.length; j++) {
                           double throughdist = distBetween(holes[i], holes[j]) +
                               holes[i].dist;
                           if (throughdist < holes[j].dist) holes[j].dist =
                               throughdist;
                        }
                  }

                  int time = (int)(10.0 * finish.dist);

                  System.out.println("Cheese " + cheeseNumber + ": Travel time =
" + time + " sec");
                  cheeseNumber++;
            } // while
      } // main

      // distance function modified for the particulars of this problem
      private static double distBetween(Hole h1, Hole h2) {
            // standard 3d distance formula
            int dx = h1.x - h2.x;
            int dy = h1.y - h2.y;
            int dz = h1.z - h2.z;

            double result = Math.sqrt((dx * dx) + (dy * dy) + (dz * dz));

            // remove radius of each hole from distance, to a minimum of 0.0
            result -= (h1.r + h2.r);
            if (result < 0.0) result = 0.0;

            return result;
      } // distBetween

      // container for info about holes in the cheese
      private static class Hole {
            public int x, y, z, r;
            public double dist;

            public Hole(int x, int y, int z) {
                  this.x = x;
                  this.y = y;
                  this.z = z;
                  this.r = 0;
            } // Hole

            public Hole(int x, int y, int z, int r) {
                  this.x = x;
                  this.y = y;
                  this.z = z;
                  this.r = r;
            } // Hole
      } // class Hole
} // class B
```

## Calculate maximal network flow (Java)

```
/* Carl Howells - Fall 2001 */
public class MaxFlow
{
	// takes an NxN adjacency matrix as first argument
	// first index is FROM vertex, second index is TO vertex
	// NO side effects.
	public static double max(double [][] capacities, int source, int target)
	{
		// create temp array
		double [][] cap = new double[capacities.length][capacities.length];

		for (int i = 0; i < cap.length; i++)
			for (int j = 0; j < cap.length; j++)
				cap[i][j] = capacities[i][j];

		double increase = 0.0;
		double result = 0.0;

		do
		{
			increase = BFS(cap, source, target);
			result += increase;
		} while (increase > 0.0);

		return result;
	} // max

	// takes an adjacency matrix as its argument
	// has a side effect on its argument, in that it
	// decreases/increases (depending on flow direction)
	// capacities by the amount of the BFS.
	private static double BFS(double [][] cap, int source, int target)
	{
		// class for items to put int the Queue
		class QItem
		{
			int vertex;
			double flow;
			QItem prev;

			public QItem(int v, double f, QItem previous)
			{
				vertex = v;
				flow = f;
				prev = previous;
			} // QItem

		} // class QItem

		// keeps track of which vertices have been visited
		boolean [] visited = new boolean[cap.length];

		java.util.LinkedList queue = new java.util.LinkedList();

		queue.add(new QItem(source, Double.MAX_VALUE, null));

		while (!queue.isEmpty())
		{
			QItem qi = (QItem)queue.getFirst();
```

```
                  // if target has been reached, jump out
                  if (qi.vertex == target) break;

                  visited[qi.vertex] = true;

                  for (int i = 0; i < visited.length; i++)
                  {
                        if (!visited[i] && cap[qi.vertex][i] > 0.0)
                        {
                              QItem toQ = new QItem(i, Math.min(qi.flow, 
cap[qi.vertex][i]), qi);
                              queue.add(toQ);
                        }
                  } // for

                  // remove current vertex from queue
                  queue.removeFirst();

            } // while

            if (queue.isEmpty())
            {
                  // didn't find any path
                  return 0.0;
            }

            QItem qi = (QItem)queue.getFirst();
            double flow = qi.flow;

            while (qi.prev != null)
            {
                  cap[qi.prev.vertex][qi.vertex] -= flow;
                  cap[qi.vertex][qi.prev.vertex] += flow;

                  qi = qi.prev;
            }

            return flow;
      } // BFS

} // class MaxFlow
```

## Factorial Frequencies – digit frequencies in factorials (Java)

```
// by Chen Zebin
// Factorial Frequencies - problem 324

import java.util.*;
import java.math.*;
import java.io.*;

public class Factor {
      public static void main(String args[]) throws Exception {
            BufferedReader d=new BufferedReader(new
InputStreamReader(System.in));
            String temp;
            Vector v=new Vector();
            Vector printV=new Vector();
            do {
                  temp=d.readLine();
                  if (temp.equals("0")) {
                        break;
                  }
                  v.addElement(temp);
                  int n=Integer.parseInt(temp);
                  printV.addElement(n+"!--");
                  String tt="\t";
                  for (int i=0; i<10; i++) {
                        int result=cal(factor(n), i);
                        tt += "(" + i + ") " + result + " ";
                                if (i==4) {
                                    tt += "\n\t";
                                }
                  }
                  printV.addElement(tt);
            } while (true);
            for (int i=0; i<printV.size(); i++) {
                  System.out.println(printV.elementAt(i));
            }
      }
      public static BigInteger factor(int n) throws Exception {
            BigInteger product=new BigInteger("1");
            for (int i=2; i<=n; i++) {
                  product=product.multiply(new BigInteger("" + i));
            }
            return product;
      }
      public static int cal(BigInteger bi, int ch) throws Exception {
            int cha=ch + '0';
            String str=bi.toString(10);
            int total=0;
            for (int i=0; i<str.length(); i++) {
                  if (str.charAt(i)==cha) {
                        total++;
                  }
            }
            return total;
      }
}
```

## Borrowers – sorting (Java)

```
/* Chen Zebin */
/* Borrowers - problem 230 */
import java.util.*;
import java.io.*;
import java.math.*;

class Book {
    public String title, author;
    public boolean in, touched;
    public Book(String title, String author) {
       this.title=title;
       this.author=author;
       in=true;
       touched=false;
    }
    public void borrow() { in=false; }
    public void sendback() { in=true; }
    public boolean largerThan(Book b2) {
       if (author.compareTo(b2.author)>0) return true;
       if (author.compareTo(b2.author)<0) return false;
       if (title.compareTo(b2.title)>0) return true;
       return false;
    }
}

public class Borrowers {
    public static void main(String args[]) throws Exception {
       BufferedReader d=new BufferedReader(new InputStreamReader(System.in));
       StringTokenizer st;
       String temp;
       Vector v=new Vector();
       Hashtable ht=new Hashtable();
       Vector printV=new Vector();

       do {
           temp=d.readLine();
           if (temp.equals("END")) break;
            int start=temp.indexOf('"', 1);
           String title=temp.substring(1, start);
            st=new StringTokenizer(temp.substring(start+2));
           String author=st.nextToken();
           author=st.nextToken();
           Book book=new Book(title, author);
           ht.put(title, book);
       } while (true);

       Book boo;
       do {
           temp=d.readLine();
           if (temp.equals("END")) break;
           if (temp.startsWith("SHELVE")) {
              Enumeration enum=ht.keys();
              int size=0;
              while (enum.hasMoreElements()) {
                  enum.nextElement();
                  size++;
              }
              Book [] books=new Book[size];
              int index=0;
              enum=ht.keys();
              while (enum.hasMoreElements()) {
                  String tempTitle=(String)enum.nextElement();
                  books[index]=(Book)ht.get(tempTitle);
                  // System.out.print(books[index].title + " ");
                  index++;
```

```
                }

                for (int i=0; i<size; i++) {
                    for (int j=1; j<size-i; j++) {
                       Book b1=(Book)books[j-1];
                       Book b2=(Book)books[j];
                       if (b1.largerThan(b2)) {
                           books[j-1]=b2;
                           books[j]=b1;
                       }
                    }
                }
//              for (int i=0; i<size; i++)  System.out.print(books[i].title + " ");
                int last=-1;
                for (int i=0; i<size; i++) {
                    boo=books[i];
                    if (boo.in && boo.touched) {
                       last=i;
                       break;
                    }
                }
                if (last<0) {
                    // System.out.println("END");
                    printV.addElement("END");
                    continue;
                } else if (last==0) {
                    // System.out.println("Put \"" + books[last].title + "\" first");
                    printV.addElement("Put \"" + books[last].title + "\" first");
                }
                while (last>0) {
                    // System.out.println("Put \"" + books[last].title + "\" after + \"" +
books[last-1].title + "\"");
                    printV.addElement("Put \"" + books[last].title + "\" after \"" +
books[last-1].title + "\"");
                    int i=last +1;
                    last=-1;
                    for (; i<size; i++) {
                       boo=books[i];
                       if (boo.in && boo.touched) {
                           last=i;
                           break;
                       }
                    }
                }
            } else {
               // System.out.println(temp);
                  int end1=temp.indexOf('"');
                  int end2=temp.indexOf('"', end1+1);
                String a=temp.substring(0, end1).trim();
                String b=temp.substring(end1+1, end2);
                boo=(Book)ht.get(b);
                if (a.equals("BORROW")) {
                    boo.borrow();
                    boo.touched=true;
                } else {
                    boo.sendback();
                    boo.touched=true;
                }
            }
        } while (true);
        for (int i=0; i<printV.size(); i++) {
                System.out.println(printV.elementAt(i));
        }
        System.out.println("END");
    }
}
```

## Borrowers – sorting (C++)

```
/* @JUDGE_ID: 14703TH 230 C++ */
/* Tim Singer - Fall 2001 */

#include <iostream.h>
#include <stdio.h>
#include <string.h>

char names[10000][81];
char auths[10000][81];
int taken[10000];
int nnames;

//Read a string from stdin
// Skip whitespace and quotes, then read until t is encountered
void readto(char* s, char t) {
    s[0]=0; int i=0; char c;
    for(c=getc(stdin); c==' ' || c=='\n' || c=='\"'; c=getc(stdin)) { }
    for(; c!=t && i<80; c=getc(stdin)) {
      s[i++]=c;
    }
    s[i]=0;
}

//selection sort
void sortnames() {
   char temp[81]; int min,tempi;
   for(int i=0; i<nnames; i++) {
      //find the name that should fit here
       min=i;
       for(int j=i; j<nnames; j++) {
           int a = strcmp(auths[min],auths[j]);
           if(a>0) min=j;
           if(a==0) {
             a=strcmp(names[min],names[j]);
             if(a>0) min=j;
           }
       }
       strcpy(temp,names[min]);      tempi=taken[min];
       strcpy(names[min],names[i]); taken[min]=taken[i];
       strcpy(names[i],temp);       taken[i]=tempi;

       strcpy(temp,auths[min]);
       strcpy(auths[min],auths[i]);
       strcpy(auths[i],temp);
   }
}

int main() {
 //mark all untaken
  for(int i=0; i<10000; i++) { taken[i]=0; auths[i][0]=names[i][0]=0; }

 //read in the book names and authors
  nnames=0;
  while(1) {
      char command[81]; char c;
      cin >> c;
      if(c=='E') {
          cin >> command; break;
      }
      readto(names[nnames],'\"');
      cin >> command; //'by'
      readto(auths[nnames],'\n');
      nnames++;
  }

 //read in commands
  while(1) {
      char command[81];
      cin >> command;
      if(!strcmp(command,"BORROW")) {
          readto(command,'\"');
         //search for the title in the list
          int j;
          for(j=0; j<nnames; j++) {
              if(!strcmp(names[j],command)) break;
          }
         //set taken flag to 1
          taken[j]=1;
      } else if(!strcmp(command,"RETURN")) {
          readto(command,'\"');
         //set taken flag to 0
         //search for the title in the list
          int j;
          for(j=0; j<nnames; j++) {
              if(!strcmp(names[j],command)) break;
          }
          if(j!=nnames) taken[j]=2;
      } else if(!strcmp(command,"END")) {
          break;
      } else if(!strcmp(command,"SHELVE")) {
          sortnames();
          int lastreturned=-1;
          for(int i=0; i<nnames; i++) {
              if(taken[i]==2) {
                  cout << "Put \"" << names[i] << "\" ";
                  if(lastreturned==-1) {
                      cout << "first" << endl;
                  } else {
                      cout << "after \"" << names[lastreturned] << "\"" << endl;
                  }
                 //return it
                  taken[i]=0; lastreturned=i;
              } else if(taken[i]==0) {
                  lastreturned=i;
              }
          }
          cout << "END" << endl;
      }
  }

  return 1;
}
```

## Factorial Frequencies – table approach (Java)

```
/* @JUDGE_ID: 324 14703TH java */
/* Arel Cordero - Fall 2001 */
import java.io.*;

class f {

    public static void main (String[] args) throws Exception {


       int[][] solution = {
           { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
           { 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
           { 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},
           { 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
           { 0, 0, 1, 0, 1, 0, 0, 0, 0, 0},
           { 1, 1, 1, 0, 0, 0, 0, 0, 0, 0},
           { 1, 0, 1, 0, 0, 0, 0, 1, 0, 0},
           .  .  .
           { 161, 67, 72, 57, 72, 69, 66, 77, 60, 70},
           { 155, 65, 83, 77, 58, 57, 73, 75, 61, 70},
           { 158, 69, 61, 74, 74, 74, 59, 70, 69, 68},
           { 166, 76, 54, 64, 72, 69, 69, 60, 70, 79},
           { 160, 93, 58, 60, 74, 81, 58, 64, 59, 74}};

       BufferedReader f = new BufferedReader ( new InputStreamReader(System.in) );
//     System.setOut(new PrintStream (new FileOutputStream ("f.out")));


       String s = f.readLine();
       while (s != "0" && s != null) {

           int i = Integer.parseInt(s);
            if(i==0) break;
           System.out.println (""+i+"! --");
           for (int x = 0; x < 10; x++) {
              System.out.print ("   ("+x+")     "+solution[i][x]);
              if (x == 4) System.out.println();
           }
           System.out.println();
           s = f.readLine();
       }

}
}
```

## Factorial Frequencies – the table generator (Java)

```
/* Arel Cordero - Fall 2001 */
import java.io.*;
import java.util.*;
import java.math.*;

class factorial {

    static char[] n = {'0','1','2','3','4','5','6','7','8','9'};

    public static void main (String [] args ) throws Exception {

      System.setOut(new PrintStream (new FileOutputStream ("factorial.out")));

      BigInteger num = new BigInteger ("1");

      for (int i = 1; i <= 366; i++) {
          num = num.multiply(new BigInteger(""+i));
          //System.err.println(num);
          int[] list = new int[10];
          String s = num.toString();
          for (int x = 0; x < 10; x ++) {
            int y = 0;
            System.err.println(s.indexOf(n[x], y));
            while (s.indexOf(n[x],y ) > -1) {
                y = s.indexOf(n[x],y)+1;
                list[x]= list[x]+1;
            }
          }
          System.out.println ("{ "+list[0]+", "+list[1]+", "+list[2]+","+list[3]
            +", "+list[4]+", "+list[5]+", "+list[6]+", "+list[7]+", "+list[8]
            +", "+list[9]+"},");
//System.err.println ("{ "+list[0]+", "+list[1]+", "+list[2]+", "+list[3]+", "
//                +list[4]+", "+list[5]+", "+list[6]+", "+list[7]+", "+list[8]
//                +", "+list[9]+"},");
      }
    }
}
```

## Triangles – find combinations in letter matrix (C)

```
/* James Marr - Fall 2001 - ACM PNW contest */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char mat[20][20];
int anumat[20][20], anlmat[20][20], apumat[20][20], aplmat[20][20], dlmat[20][20],
dtmat[20][20], drmat[20][20], dbmat[20][20];
int N;

#define true 1
#define false 0

int match (char c, int x, int y) {
        if ((x>=0) && (x < N) && (y>=0) && (y<N) && (mat[x][y]==c))
                return true;
        else
                return false;
}

#define adef(amat,dx,dy,sx,sy,compx,compy) {  \
        int x,y; \
        memset (amat, 0, sizeof amat); \
        for (x=sx;x compx; x -= dx) { \
                for (y=sy;y compy; y -= dy) { \
                        char c= mat[x][y]; \
                        if (match (c, x+dx, y) && match (c,x,y+dy)) { \
                                if (amat[x+dx][y] < amat[x][y+dy]) \
                                        amat[x][y] = 1 + amat[x+dx][y]; \
                                else \
                                        amat[x][y] = 1 + amat[x][y+dy]; \
                        } \
                } \
        } \
}

#define ddef(dmat,x1,y1,x2,y2,x3,y3,m2,m3,name) { \
        int x,y; \
        memset (dmat, 0, sizeof dmat); \
        for (x=0; x < N; x++) { \
                for (y = 0; y<N; y++) { \
                        char c = mat[x][y]; \
                        int span, i; \
                        if (match (c, x+x2, y+y2) && match (c, x+x3, y+y3)) { \
                                if (m2[x+x2][y+y2] < m3[x+x3][y+y3]) \
                                        span = m2[x+x2][y+y2] + 1; \
                                else \
                                        span = m3[x+x3][y+y3] + 1; \
                                for (i=1; i <=span; i++) { \
                                        if (match(c, x+x1*i, y+y1*i)) \
                                                dmat[x][y]++; \
                                        else \
                                                goto name; \
                                } \
                        } \
          name: \
                } \
        } \
}

#define debugify(array) \
                { \
                        printf ("%s:\n", #array); \
                        for (i=0;i<N;i++) { \
                                for (j=0;j<N;j++) { \
                                        printf ("%d", array[i][j]); \
                                } \
                                printf ("\n"); \
                        } \
                }

int main (void) {
        FILE *f = fopen ("C.in", "r");
        char line[256];
        int i, j;

        while (1) {
                fgets (line, sizeof line, f);
                N = strtoul (line, NULL, 10);
                if (!N) break;
                for (i=0; i< N; i++) {
                        fgets (line, sizeof line, f);
                        for (j=0; j< N;j++) {
                                mat[j][i] = line[j];
                        }
                }

                adef (anumat, -1, 1, 0, N-1, <N, >=0);
                adef (anlmat, 1, -1, N-1, 0, >=0, <N);
                adef (apumat, 1, 1, N-1, N-1, >=0, >=0);
                adef (aplmat, -1, -1, 0, 0, <N, <N);

                ddef (dlmat, 1, 0, 0, -1, 0, 1, anlmat, apumat, dlmatbreak);
                ddef (dtmat, 0, 1, -1, 0, 1, 0, anumat, apumat, dtmatbreak);
                ddef (drmat, -1, 0, 0, -1, 0, 1, aplmat, anumat, drmatbreak);
                ddef (dbmat, 0, -1, -1, 0, 1, 0, aplmat, anlmat, dbmatbreak);

                /* debugify(anumat); debugify(anlmat); debugify(apumat);
                debugify(aplmat); debugify(dlmat); debugify(dtmat);
                debugify(drmat); debugify(dbmat);*/

                {
                        int total = 0, letters[256], bletters[256], partial;

                        memset (letters, 0, sizeof (letters));
                        memset (bletters, 0, sizeof (letters));

                        for (i=0;i < N; i++) {
                                for (j=0;j<N; j++) {
                                        partial = anumat[i][j];
                                        partial += anlmat[i][j];
                                        partial += apumat[i][j];
                                        partial += aplmat[i][j];
                                        partial += dlmat[i][j];
                                        partial += dtmat[i][j];
                                        partial += drmat[i][j];
                                        partial += dbmat[i][j];

                                        total += partial;
                                        letters[mat[i][j]] += partial;
                                        bletters[mat[i][j]] = 1;
                                }
                        }
                        printf ("(%d)", total);

                        for (i=0; i< 256; i++) {
                                if (!bletters[i]) continue;
                                printf (" %d %c", letters[i], i);
                        }
                        printf ("\n");
                }
        }
        return 0;
}
```

## Tax Relief – calculate tax rebates (C)

```
/* James Marr - Fall 2001 - ACM PNW contest */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct week
{
        char  name[256];
        int numpeople;
        float money;
}week;

week weeks[10] = {{"July 23", 0, 0},
        {"July 30", 0, 0},
        {"August 6", 0, 0},
        {"August 13", 0, 0},
        {"August 20", 0, 0},
        {"August 27", 0, 0},
        {"September 3", 0, 0},
        {"September 10", 0, 0},
        {"September 17", 0, 0},
        {"September 24", 0, 0}};

int main(void)
{
        FILE *f;
        int i;
        char buf[256];
        char *s;

        f = fopen("E.in", "r");

        while(1)
        {
                char ss[256];
                int type;
                float income, liability, min;
                fgets(buf, sizeof(buf), f);
                s = buf;
                if(!strcmp(s, "000-00-0000\n")) break;
                *strchr(s, ' ') = 0;
                strcpy(ss, s);
                s += strlen(s) + 1;

                type = strtol(s, &s, 10);
                income = strtod(s, &s);
                liability = strtod(s, &s);

                income = income * 0.05;

                switch(type)
                {
                case 1:
                case 3:
                        min = 300;
                        break;
                case 2:
                case 5:
                        min = 600;
                        break;
                case 4:
                        min = 500;
                        break;
                }
                if(income < min) min = income;
                if(liability < min) min = liability;
                printf("%s  $%.2f\n", ss, min);
                s = &ss[9];
                ss[10] = 0;
                type = strtol(s, 0, 10);

                sprintf (buf, "%.2f", min);
                min = strtod (buf, NULL);

                weeks[type].numpeople++;
                weeks[type].money += min;
        }
        for(i=0; i<10; i++)
        {
                if(weeks[i].numpeople > 0)
                        printf("%d  $%.2f %s\n", weeks[i].numpeople,
weeks[i].money, &weeks[i].name);
        }
        return 0;
}
```