answers10

1.

```c
#include <stdio.h>
#define MAX 100

bool isLeader(int data[], int n, int k);
int main()
{
    int data[MAX];
    int n = 0, k;
    FILE *fp = fopen("input10_1.txt", "r");
    while(!feof(fp))  {
        fscanf(fp, "%d", &k);
        if (isLeader(data, n, k))
            data[n++] = k;
    }
    fclose(fp);
    printf("%d: ", n);
    for (int i=0; i<n; i++)
        printf("%d ", data[i]);
}

//bool isLeader(int data[], int n, int k) {
//    for (int i=0; i<n; i++)
//        if (data[i] > k)
//            return false;
//    return true;
//}

bool isLeader(int data[], int n, int k) {
    return  (n == 0 || data[n-1] <= k);
}
```

2.

```c
#include <stdio.h>
#define MAX 100
int find(int data[], int n, int k);
void insert(int data[], int n, int k);

int main()
{
    int data[MAX];
    int n = 0, k;
    while(1) {
        scanf("%d", &k);
        if (k==-1) break;
        int index = find(data, n, k);
        if (index != -1)
            printf("duplicate entry: %d\n", index);
        else {
            insert(data, n, k);
            n++;
            for (int i=0; i<n; i++)
                printf("%d ", data[i]);
            printf("\n");
        }
    }
}
```

```
void insert(int data[], int n, int k) {
    int i=n-1;
    while(i>=0 && data[i] > k) {
        data[i+1] = data[i];
        i--;
    }
    data[i+1] = k;
}

int find(int data[], int n, int k) {
    for (int i=0; i<n; i++)
        if (data[i] == k)
            return i;
    return -1;
}
```

3.

```
#include <stdio.h>
#define MAX 100

bool intersect( int p, int q, int x1[], int y1[], int x2[], int y2[]);
void insertion_sort(int n, int xx[], int yy[])
int findAllIntersections(int n, int x1[], int y1[], int x2[],
                                    int y2[], int xx[], int yy[]);

int main()
{
    int x1[MAX], y1[MAX], x2[MAX], y2[MAX];
    int xx[MAX], yy[MAX];
    int n = 0;

    FILE *fd = fopen("input10_3.txt", "r");
    while(!feof(fd)) {
        fscanf(fd, "%d %d %d %d", &x1[n], &y1[n], &x2[n], &y2[n]);
        n++;
    }
    fclose(fd);

    int count = findAllIntersections( n, x1, y1, x2, y2, xx, yy );
    insertion_sort( count, xx, yy );

    for (int i=0; i<count; i++)
        printf("[%d, %d]\n", xx[i], yy[i]);
}

int findAllIntersections(int n, int x1[], int y1[], int x2[], int y2[],
                                int xx[], int yy[]) {
    int count = 0;
    for (int i=0; i<n; i++) {
        for (int j=i+1; j<n; j++) {
            if (!intersect( i, j, x1, y1, x2, y2)) continue;
            if (x1[i] == x2[i]) {   // i  vertical and j horizontal
                xx[count] = x1[i];
                yy[count++] = y1[j];
            }
            else  {  // i horizontal and j vertical
                xx[count] = x1[j];
                yy[count++] = y1[i];
            }
```

```
        }
    }
    return count;
}

/* decide whether p-th and q-th line segments intersect */
bool intersect( int p, int q, int x1[], int y1[], int x2[], int y2[])
{
    if (x1[p] == x2[p] && x1[q] == x2[q] || y1[p] == y2[p]
                && y1[q] == y2[q])  //  both horizontal or both vertical
        return false;
    if (x1[p] == x2[p] && x1[q] <= x1[p] && x2[q] >= x1[p] && y1[p] <= y1[q]
                && y2[p] >= y1[q] )   // p is vertical
        return true;
    else if (x1[p] <= x1[q] && x2[p] >= x1[q] && y1[q] <= y1[p]
                && y2[q] >= y1[p])
        return true;
    return false;
}

void insertion_sort(int count, int xx[], int yy[])
{
    for (int i=1; i<count; i++) {
        int tmpx = xx[i];
        int tmpy = yy[i];
        int j = i-1;
        while ( j>=0 && (xx[j] > tmpx || xx[j] == tmpx && yy[j] > tmpy )) {
            xx[j+1] = xx[j];
            yy[j+1] = yy[j];
            j--;
        }
        xx[j+1] = tmpx;
        yy[j+1] = tmpy;
    }
}

4.
#include <stdio.h>
#define MAX 2000
int merge(int n1, int data1[], int n2, int data2[], int data3[]);
int main() {
    int data1[MAX], data2[MAX], data3[MAX];
    int n1 = 0, n2 = 0;
    FILE *fp1 = fopen("input4-1.txt", "r");
    FILE *fp2 = fopen("input4-2.txt", "r");
    while (!feof(fp1))  fscanf(fp1, "%d", &data1[n1++]);
    while (!feof(fp2))  fscanf(fp2, "%d", &data2[n2++]);
    fclose(fp1); fclose(fp2);

    int n3 = merge( n1, data1, n2, data2, data3);
    for (int i=0; i<n3; i++)
        printf("%d\n", data3[i]);
}

int merge(int n1, int data1[], int n2, int data2[], int data3[]) {
    int i = 0, j = 0, k = 0;
    while( i < n1 && j < n2) {
        if (data1[i] <= data2[j])
            data3[k++] = data1[i++];
        else
```

```
            data3[k++] = data2[j++];
    }
    while(i<n1) data3[k++] = data1[i++];
    while(j<n2) data3[k++] = data2[j++];
    return k;
}
```

5.

```
#include <stdio.h>
#define MAX 1000
bool check(int n, int traj[][2], int x, int y);
bool intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4);

int main()
{
    int offset[][2] = { {0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    int n = 1;
    int trajectory[MAX][2];   // trajectory
    int dir, dist, lastDir = -1;
    int x = 0, y = 0;    // current position
    trajectory[0][0] = 0, trajectory[0][1] = 0;

    while(1) {
        scanf("%d %d", &dir, &dist);
        if (dir == -1) break;

        int nextX = x + dist*offset[dir][0];
        int nextY = y + dist*offset[dir][1];

        if (lastDir!=-1 && (dir+2)%4==lastDir
                        || !check(n, trajectory, nextX, nextY))
            printf("invalid move\n");
        else {
            trajectory[n][0] = x = nextX;
            trajectory[n++][1] = y = nextY;
            lastDir = dir;
            printf("%d %d\n", x, y);
        }
    }
}

bool check(int n, int traj[][2], int x, int y) {
    for (int i=1; i<n-1; i++)
        if (intersect(traj[i-1][0], traj[i-1][1], traj[i][0],
                            traj[i][1], traj[n-1][0], traj[n-1][1], x, y))
            return false;
    return true;
}

bool intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
    bool vert12 = (x1 == x2), vert34 = (x3 == x4);

    if (vert12 && y1 > y2) {
        int tmp = y1; y1 = y2; y2 = tmp;
    }
    else if (!vert12 && x1 > x2) {
        int tmp = x1; x1 = x2; x2 = tmp;
    }
    if (vert34 && y3 > y4) {
        int tmp = y3; y3 = y4; y4 = tmp;
```

```
    }
    else if (!vert34 && x3 > x4) {
        int tmp = x3; x3 = x4; x4 = tmp;
    }

    if (vert12 && vert34)
        return !(x1 != x3 || y2 < y3 || y4 < y1);
    else if (vert12 && !vert34)
        return  (x3 <= x1 && x1 <= x4 && y1 <= y3 && y3 <= y2);
    else if (!vert12 && vert34)
        return (x1 <= x3 && x3 <= x2 && y3 <= y1 && y1 <= y4);
    else
        return !(y1 != y3 || x2 < x3 || x4 < x1);
}
```