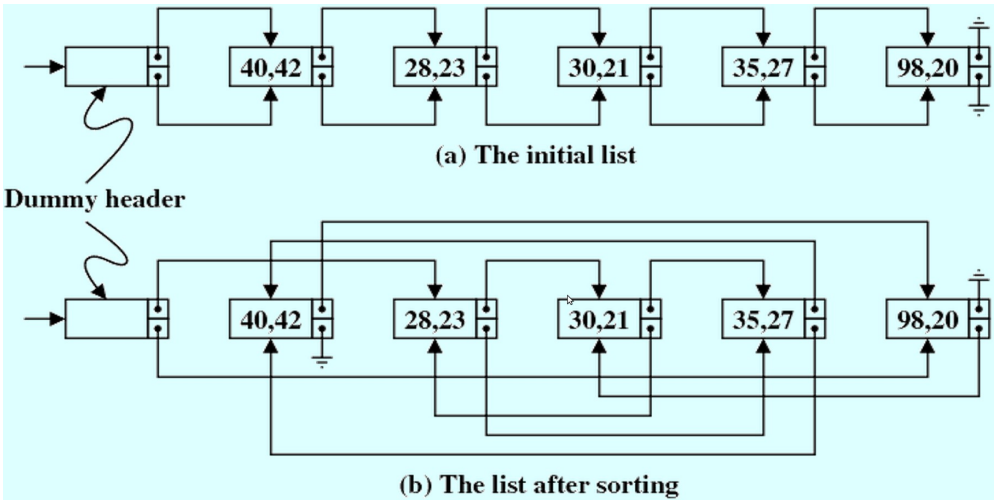
INDIAN INSTITUTE OF TECHNOLOGY MADRAS
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Name: ***Programming and Data Structure Lab*** **Course No:** ***CS2710***
Full Marks: 15 (LabTest-1) Date: 19 / 09 / 2016 Time: 3 *hours*

Instructions:
You are supplied with two files: `ROLLNO_labtest1_prob1.c` and `ROLLNO_labtest1_prob2.c`. First, replace these file-names with your Roll-Numbers where the `ROLLNO` keyword appears in the file-name. You can ***only modify*** the mentioned areas (where it is denoted as "`// Write C-code Here`") in these two `.c` files. You are ***not allowed*** to modify/change any other portion of the given code. Please make sure that your programs compile and run successfully. Finally, submit only two `.c` files (i.e., `ROLLNO_labtest1_prob1.c` and `ROLLNO_labtest1_prob2.c`) in the mentioned submission links given in MOODLE.

Problem-1: **[7 *Marks*]**

Suppose that we have a linked list of points in the X-Y plane. We want to sort the list simultaneously with respect the X-coordinates and with respect to the Y-coordinates. To achieve this goal, we maintain two pointers in each node. One set of these pointers is used to sort the list with respect to the X-values, the other with respect to the Y-values. The following figure demonstrates this concept.



In view of this representation, we declare some relevant data types as follows.

```
typedef struct _node {
     int x;
     int y;
     struct _node *nextx;
     struct _node *nexty;
} node;
typedef node *list;
```

First, generate a random list of points with integer coordinates. At this point, you do not worry about sorting of the elements, but insert every new element at the end of the new list with respect to both the `nextx` and `nexty` pointers. A function to do that is given below.

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

list genRandList ( int n )
{
```

```
        list L;
        node *p, *q;
        int i;
        /* Create dummy header */
        L = (list)malloc(sizeof(node));
        L -> x = L -> y = 0;
        L -> nextx = L -> nexty = NULL;
        /* Insert random elements at the end of the list */
        srand((unsigned int)time(NULL));
        p = L;
        for (i=0; i<n; ++i) {
             q = (node *)malloc(sizeof(node));
             q -> x = 1000 + rand() % 9000;
             q -> y = 1000 + rand() % 9000;
             q -> nextx = q -> nexty = NULL;
             p -> nextx = p -> nexty = q;
             p = q;
        }
        return L;
}
```

Complete the C-function with the following prototype:

```
void printList ( list L, int flag );
```

This function prints the list headed by the pointer L. Now, each node has two pointers. The flag indicates whether the list is to be traversed along the `nextx` pointers, or along the `nexty` pointers.

Complete the two C-functions with the following prototypes:

```
void bubbleSortX ( list L );
void bubbleSortY ( list L );
```

The first function is meant for bubble-sorting the list headed by L with respect to the x-values, and the second with respect to the y-values. You are not supposed to separate the x and y coordinates of a point. This means that a swap in only the x-values or only the y-values in two nodes is not permitted. If you swap both the x and y values together, you disturb the order in the values with respect to which you are not sorting. So you must *adjust the relevant pointers* in the nodes in order to effect two independent sorting of the same list.

Your functions should be compatible with the following `main()` function.

```
#define N 100
#define WRT_X 0
#define WRT_Y 1
int main ()
{
        list L;
        L = genRandList(N);
        printf("\nInitial list with respect to x pointers:\n");
        printList(L,WRT_X);
        printf("\nInitial list with respect to y pointers:\n");
        printList(L,WRT_Y);
        bubbleSortX(L);
        bubbleSortY(L);
        printf("\nFinal list with respect to x pointers:\n");
        printList(L,WRT_X);
        printf("\nFinal list with respect to y pointers:\n");
        printList(L,WRT_Y);
        return(0);
}
```

Report the output of your program for a random list of 100 points.

Sample Output:

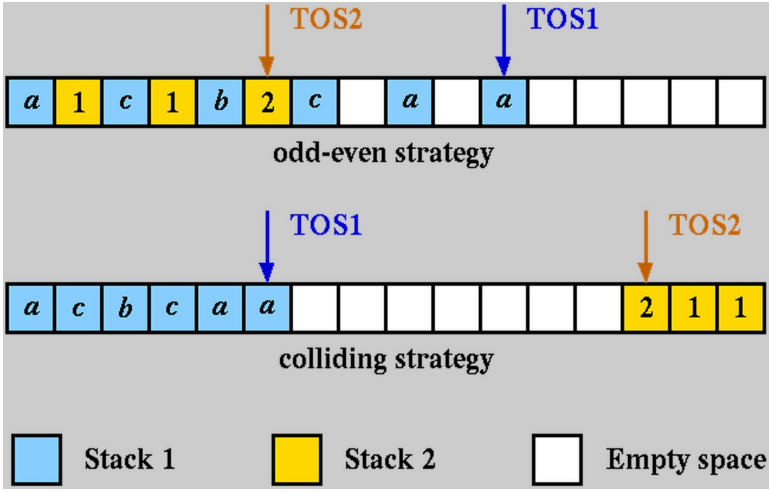
```
Initial list with respect to x pointers:
(8819,9618) (8589,1044) (9705,8523) (6264,9407) (8792,4613) (8090,1571)
(7630,5719) (4810,4851) (8927,3233) (7097,2454) (9693,8946) (7902,7518)
(4484,1503) (7529,8466) (4996,6951) (6408,3815) (6569,4997) (3859,3627)
(3520,6475) (3034,8665) (7441,1125) (6588,2423) (3196,1398) (6275,8476)
(9983,9724) (9930,9677) (6022,5185) (4547,9506) (5688,8428) (5324,7036)
(5380,1732) (9851,8301) (3081,1062) (1928,5601) (6537,1315) (4266,1330)
(1440,9854) (2754,9988) (7604,5381) (8464,7588) (5105,5747) (4617,1127)
(9932,5516) (6985,2972) (3944,8661) (9008,5676) (9393,6211) (3978,2474)
(3625,2258) (4427,9162) (2573,5046) (6845,9365) (2252,8599) (6706,8857)
(3980,2522) (3797,5437) (7269,7414) (2916,4553) (2930,6253) (6525,3226)
(4914,2885) (7903,4307) (5448,8233) (3133,8073) (6843,6560) (4588,5769)
(7958,1433) (2486,9211) (6384,8192) (5420,9364) (9715,8217) (2153,4336)
(2983,4069) (7890,2265) (6674,1767) (1843,1588) (3653,8746) (2247,5453)
(4331,4380) (3527,7527) (7292,7115) (3296,5251) (4900,2134) (1814,1284)
(6679,6234) (7000,6394) (1803,8153) (7082,3786) (8574,4972) (2403,5248)
(5740,3246) (3188,5745) (8345,4435) (1198,2676) (5167,1077) (6555,2459)
(4544,8851) (4062,8444) (7338,4876) (8728,4017)

Initial list with respect to y pointers:
(8819,9618) (8589,1044) (9705,8523) (6264,9407) (8792,4613) (8090,1571)
(7630,5719) (4810,4851) (8927,3233) (7097,2454) (9693,8946) (7902,7518)
(4484,1503) (7529,8466) (4996,6951) (6408,3815) (6569,4997) (3859,3627)
(3520,6475) (3034,8665) (7441,1125) (6588,2423) (3196,1398) (6275,8476)
(9983,9724) (9930,9677) (6022,5185) (4547,9506) (5688,8428) (5324,7036)
(5380,1732) (9851,8301) (3081,1062) (1928,5601) (6537,1315) (4266,1330)
(1440,9854) (2754,9988) (7604,5381) (8464,7588) (5105,5747) (4617,1127)
(9932,5516) (6985,2972) (3944,8661) (9008,5676) (9393,6211) (3978,2474)
(3625,2258) (4427,9162) (2573,5046) (6845,9365) (2252,8599) (6706,8857)
(3980,2522) (3797,5437) (7269,7414) (2916,4553) (2930,6253) (6525,3226)
(4914,2885) (7903,4307) (5448,8233) (3133,8073) (6843,6560) (4588,5769)
(7958,1433) (2486,9211) (6384,8192) (5420,9364) (9715,8217) (2153,4336)
(2983,4069) (7890,2265) (6674,1767) (1843,1588) (3653,8746) (2247,5453)
(4331,4380) (3527,7527) (7292,7115) (3296,5251) (4900,2134) (1814,1284)
(6679,6234) (7000,6394) (1803,8153) (7082,3786) (8574,4972) (2403,5248)
(5740,3246) (3188,5745) (8345,4435) (1198,2676) (5167,1077) (6555,2459)
(4544,8851) (4062,8444) (7338,4876) (8728,4017)

Final list with respect to x pointers:
(1198,2676) (1440,9854) (1803,8153) (1814,1284) (1843,1588) (1928,5601)
(2153,4336) (2247,5453) (2252,8599) (2403,5248) (2486,9211) (2573,5046)
(2754,9988) (2916,4553) (2930,6253) (2983,4069) (3034,8665) (3081,1062)
(3133,8073) (3188,5745) (3196,1398) (3296,5251) (3520,6475) (3527,7527)
(3625,2258) (3653,8746) (3797,5437) (3859,3627) (3944,8661) (3978,2474)
(3980,2522) (4062,8444) (4266,1330) (4331,4380) (4427,9162) (4484,1503)
(4544,8851) (4547,9506) (4588,5769) (4617,1127) (4810,4851) (4900,2134)
(4914,2885) (4996,6951) (5105,5747) (5167,1077) (5324,7036) (5380,1732)
(5420,9364) (5448,8233) (5688,8428) (5740,3246) (6022,5185) (6264,9407)
(6275,8476) (6384,8192) (6408,3815) (6525,3226) (6537,1315) (6555,2459)
(6569,4997) (6588,2423) (6674,1767) (6679,6234) (6706,8857) (6843,6560)
(6845,9365) (6985,2972) (7000,6394) (7082,3786) (7097,2454) (7269,7414)
(7292,7115) (7338,4876) (7441,1125) (7529,8466) (7604,5381) (7630,5719)
(7890,2265) (7902,7518) (7903,4307) (7958,1433) (8090,1571) (8345,4435)
(8464,7588) (8574,4972) (8589,1044) (8728,4017) (8792,4613) (8819,9618)
(8927,3233) (9008,5676) (9393,6211) (9693,8946) (9705,8523) (9715,8217)
(9851,8301) (9930,9677) (9932,5516) (9983,9724)

Final list with respect to y pointers:
(8589,1044) (3081,1062) (5167,1077) (7441,1125) (4617,1127) (1814,1284)
(6537,1315) (4266,1330) (3196,1398) (7958,1433) (4484,1503) (8090,1571)
(1843,1588) (5380,1732) (6674,1767) (4900,2134) (3625,2258) (7890,2265)
(6588,2423) (7097,2454) (6555,2459) (3978,2474) (3980,2522) (1198,2676)
(4914,2885) (6985,2972) (6525,3226) (8927,3233) (5740,3246) (3859,3627)
(7082,3786) (6408,3815) (8728,4017) (2983,4069) (7903,4307) (2153,4336)
(4331,4380) (8345,4435) (2916,4553) (8792,4613) (4810,4851) (7338,4876)
(8574,4972) (6569,4997) (2573,5046) (6022,5185) (2403,5248) (3296,5251)
(7604,5381) (3797,5437) (2247,5453) (9932,5516) (1928,5601) (9008,5676)
(7630,5719) (3188,5745) (5105,5747) (4588,5769) (9393,6211) (6679,6234)
(2930,6253) (7000,6394) (3520,6475) (6843,6560) (4996,6951) (5324,7036)
(7292,7115) (7269,7414) (7902,7518) (3527,7527) (8464,7588) (3133,8073)
(1803,8153) (6384,8192) (9715,8217) (5448,8233) (9851,8301) (5688,8428)
(4062,8444) (7529,8466) (6275,8476) (9705,8523) (2252,8599) (3944,8661)
(3034,8665) (3653,8746) (4544,8851) (6706,8857) (9693,8946) (4427,9162)
(2486,9211) (5420,9364) (6845,9365) (6264,9407) (4547,9506) (8819,9618)
(9930,9677) (9983,9724) (1440,9854) (2754,9988)
```

Problem-2: **[8 *Marks*]**

Suppose you want to implement two stacks using a single array. Two possibilities are outlined here:

- **Odd-even strategy:** In this case, Stack 1 uses locations 0, 2, 4, ... of the array, whereas Stack 2 uses the array locations 1, 3, 5, ...
- **Colliding strategy:** In this case, the two stacks start from the two ends of the array and grow in opposite directions (towards one another).

The following figure demonstrates these two strategies.



Implement both the strategies. Write two sets of initialize, push and pop routines corresponding to the two strategies. ***Complete*** the following three C-functions with the prototypes given below.

- Initialize Stack: `void initStack ( int option );`
- Push into Stack: `void pushStack ( int which , char what , int option );`
- Pop from Stack: `void popStack ( int which , int option );`

Please note that, your code should also print `Overflow-Error` (from `pushStack` function) and `Underflow-Error` (from `popStack` function) for both the stacks.

Show the output of your code on randomly generated sequences of requests. In order to have good-looking outputs, you may enforce the following:

- Keep the array size small, say 32 (max).
- For the first few iterations do push only.
- Subsequently make push twice as likely as pop.
- Make the first stack twice more active (on an average) than the second.

You may rely on the randomness of C's built-in pseudorandom generator to ensure these statistical properties. Your functions should be compatible with the following `main()` function.

```
#include <stdio.h>
#include<stdlib.h>
#define MAX_SIZE 32

char store[MAX_SIZE];
int tos1, tos2;

int main ()
{
```

```
        int i, option, which, action;
        char what;

        srand((unsigned int)time(NULL));

        printf("Enter strategy -- 0 (odd-even) or 1 (colliding) : ");
        scanf("%d",&option);
        initStack(option);

        i = 0;
        while (1) {
             ++i;
             printf("Iteration %3d : ", i); fflush(stdout);

             /* Initially make push */
             /* Then make push twice as likely as pop */
             if (i <= 8) action = 1;
             else action = rand() % 3;

             /* Make Stack 1 twice more active than Stack 2 */
             which = (rand() % 3) ? 1 : 2;

             if (action) {    /* Push */
                  what = ((which == 1) ? 'a' : 'A') + (rand() % 26);
                  printf("Push %c in stack %d. New stack : ",what,which);
                  fflush(stdout);
                  pushStack(which,what,option);
             }
             else {           /* Pop */
                  printf("Pop    in stack %d. New stack : ",which);
                  fflush(stdout);
                  popStack(which,option);
             }
             printStack();
        }
        return(0);
   }
```

Don't print very big output files. Generate 4 runs of your program with a total size of less than 300 lines. Two runs should correspond to the odd-even strategy, the remaining two the colliding strategy.

Sample Output:

```
Enter strategy -- 0 (odd-even) or 1 (colliding) : 0
Iteration   1 : Push p in stack 1. New stack : p_____________________________
Iteration   2 : Push i in stack 1. New stack : p_i___________________________
Iteration   3 : Push o in stack 1. New stack : p_i_o_________________________
Iteration   4 : Push x in stack 1. New stack : p_i_o_x_______________________
Iteration   5 : Push z in stack 1. New stack : p_i_o_x_z_____________________
Iteration   6 : Push r in stack 1. New stack : p_i_o_x_z_r___________________
Iteration   7 : Push n in stack 1. New stack : p_i_o_x_z_r_n_________________
Iteration   8 : Push f in stack 1. New stack : p_i_o_x_z_r_n_f_______________
Iteration   9 : Pop    in stack 1. New stack : p_i_o_x_z_r_n_________________
Iteration  10 : Push s in stack 1. New stack : p_i_o_x_z_r_n_s_______________
Iteration  11 : Push j in stack 1. New stack : p_i_o_x_z_r_n_s_j_____________
Iteration  12 : Push W in stack 2. New stack : pWi_o_x_z_r_n_s_j_____________
Iteration  13 : Pop    in stack 1. New stack : pWi_o_x_z_r_n_s_______________
Iteration  14 : Push X in stack 2. New stack : pWiXo_x_z_r_n_s_______________
Iteration  15 : Push g in stack 1. New stack : pWiXo_x_z_r_n_s_g_____________
Iteration  16 : Push x in stack 1. New stack : pWiXo_x_z_r_n_s_g_x___________
Iteration  17 : Pop    in stack 2. New stack : pWi_o_x_z_r_n_s_g_x___________
Iteration  18 : Pop    in stack 1. New stack : pWi_o_x_z_r_n_s_g_____________
Iteration  19 : Push t in stack 1. New stack : pWi_o_x_z_r_n_s_g_t___________
Iteration  20 : Pop    in stack 1. New stack : pWi_o_x_z_r_n_s_g_____________
Iteration  21 : Push e in stack 1. New stack : pWi_o_x_z_r_n_s_g_e___________
Iteration  22 : Pop    in stack 1. New stack : pWi_o_x_z_r_n_s_g_____________
```

```
Iteration  23 : Push R in stack 2. New stack : pWiRo_x_z_r_n_s_g_________________
Iteration  24 : Push p in stack 1. New stack : pWiRo_x_z_r_n_s_g_p_______________
Iteration  25 : Push I in stack 2. New stack : pWiRoIx_z_r_n_s_g_p_______________
Iteration  26 : Push r in stack 1. New stack : pWiRoIx_z_r_n_s_g_p_r_____________
Iteration  27 : Push h in stack 1. New stack : pWiRoIx_z_r_n_s_g_p_r_h___________
Iteration  28 : Push u in stack 1. New stack : pWiRoIx_z_r_n_s_g_p_r_h_u_________
Iteration  29 : Push C in stack 2. New stack : pWiRoIxCz_r_n_s_g_p_r_h_u_________
Iteration  30 : Push v in stack 1. New stack : pWiRoIxCz_r_n_s_g_p_r_h_u_v_______
Iteration  31 : Pop    in stack 1. New stack : pWiRoIxCz_r_n_s_g_p_r_h_u_________
Iteration  32 : Pop    in stack 1. New stack : pWiRoIxCz_r_n_s_g_p_r_h___________
Iteration  33 : Push f in stack 1. New stack : pWiRoIxCz_r_n_s_g_p_r_h_f_________
Iteration  34 : Push Y in stack 2. New stack : pWiRoIxCzYr_n_s_g_p_r_h_f_________
Iteration  35 : Push n in stack 1. New stack : pWiRoIxCzYr_n_s_g_p_r_h_f_n_______
Iteration  36 : Pop    in stack 1. New stack : pWiRoIxCzYr_n_s_g_p_r_h_f_________
Iteration  37 : Push D in stack 2. New stack : pWiRoIxCzYrDn_s_g_p_r_h_f_________
Iteration  38 : Push t in stack 1. New stack : pWiRoIxCzYrDn_s_g_p_r_h_f_t_______
Iteration  39 : Push E in stack 2. New stack : pWiRoIxCzYrDnEs_g_p_r_h_f_t_______
Iteration  40 : Push O in stack 2. New stack : pWiRoIxCzYrDnEsOg_p_r_h_f_t_______
Iteration  41 : Push x in stack 1. New stack : pWiRoIxCzYrDnEsOg_p_r_h_f_t_x_____
Iteration  42 : Push k in stack 1. New stack : pWiRoIxCzYrDnEsOg_p_r_h_f_t_x_k___
Iteration  43 : Push f in stack 1. New stack : Error: Overflow in Stack 1.

Enter strategy -- 0 (odd-even) or 1 (colliding) : 0
Iteration   1 : Push p in stack 1. New stack : p_______________________________
Iteration   2 : Push w in stack 1. New stack : p_w_____________________________
Iteration   3 : Push i in stack 1. New stack : p_w_i___________________________
Iteration   4 : Push m in stack 1. New stack : p_w_i_m_________________________
Iteration   5 : Push q in stack 1. New stack : p_w_i_m_q_______________________
Iteration   6 : Push i in stack 1. New stack : p_w_i_m_q_i_____________________
Iteration   7 : Push d in stack 1. New stack : p_w_i_m_q_i_d___________________
Iteration   8 : Push l in stack 1. New stack : p_w_i_m_q_i_d_l_________________
Iteration   9 : Pop    in stack 2. New stack : Error: Underflow in Stack 2.

Enter strategy -- 0 (odd-even) or 1 (colliding) : 1
Iteration   1 : Push j in stack 1. New stack : j_______________________________
Iteration   2 : Push x in stack 1. New stack : jx______________________________
Iteration   3 : Push U in stack 2. New stack : jx_____________________________U
Iteration   4 : Push r in stack 1. New stack : jxr____________________________U
Iteration   5 : Push K in stack 2. New stack : jxr____________________________KU
Iteration   6 : Push z in stack 1. New stack : jxrz___________________________KU
Iteration   7 : Push m in stack 1. New stack : jxrzm__________________________KU
Iteration   8 : Push b in stack 1. New stack : jxrzmb_________________________KU
Iteration   9 : Pop    in stack 2. New stack : jxrzmb_________________________U
Iteration  10 : Pop    in stack 1. New stack : jxrzm__________________________U
Iteration  11 : Pop    in stack 2. New stack : jxrzm__________________________
Iteration  12 : Pop    in stack 1. New stack : jxrz___________________________
Iteration  13 : Pop    in stack 1. New stack : jxr____________________________
Iteration  14 : Push n in stack 1. New stack : jxrn___________________________
Iteration  15 : Push D in stack 2. New stack : jxrn__________________________D
Iteration  16 : Push c in stack 1. New stack : jxrnc_________________________D
Iteration  17 : Push H in stack 2. New stack : jxrnc________________________HD
Iteration  18 : Pop    in stack 1. New stack : jxrn________________________HD
Iteration  19 : Push G in stack 2. New stack : jxrn_______________________GHD
Iteration  20 : Push f in stack 1. New stack : jxrnf______________________GHD
Iteration  21 : Pop    in stack 1. New stack : jxrn_______________________GHD
Iteration  22 : Push V in stack 2. New stack : jxrn______________________VGHD
Iteration  23 : Push f in stack 1. New stack : jxrnf_____________________VGHD
Iteration  24 : Pop    in stack 2. New stack : jxrnf_____________________GHD
Iteration  25 : Push n in stack 1. New stack : jxrnfn____________________GHD
Iteration  26 : Pop    in stack 2. New stack : jxrnfn____________________HD
Iteration  27 : Pop    in stack 1. New stack : jxrnf_____________________HD
Iteration  28 : Push z in stack 1. New stack : jxrnfz____________________HD
Iteration  29 : Push G in stack 2. New stack : jxrnfz____________________GHD
Iteration  30 : Push a in stack 1. New stack : jxrnfza___________________GHD
Iteration  31 : Push X in stack 2. New stack : jxrnfza__________________XGHD
Iteration  32 : Push w in stack 1. New stack : jxrnfzaw_________________XGHD
Iteration  33 : Push L in stack 2. New stack : jxrnfzaw________________LXGHD
Iteration  34 : Push S in stack 2. New stack : jxrnfzaw_______________SLXGHD
Iteration  35 : Push q in stack 1. New stack : jxrnfzawq______________SLXGHD
Iteration  36 : Pop    in stack 1. New stack : jxrnfzaw______________SLXGHD
Iteration  37 : Push m in stack 1. New stack : jxrnfzawm_____________SLXGHD
Iteration  38 : Push Y in stack 2. New stack : jxrnfzawm____________YSLXGHD
Iteration  39 : Push z in stack 1. New stack : jxrnfzawmz___________YSLXGHD
Iteration  40 : Push h in stack 1. New stack : jxrnfzawmzh__________YSLXGHD
Iteration  41 : Push y in stack 1. New stack : jxrnfzawmzhy_________YSLXGHD
Iteration  42 : Push N in stack 2. New stack : jxrnfzawmzhy________NYSLXGHD
Iteration  43 : Pop    in stack 1. New stack : jxrnfzawmzh________NYSLXGHD
Iteration  44 : Push i in stack 1. New stack : jxrnfzawmzhi_______NYSLXGHD
```

```
Iteration  45 : Push c in stack 1. New stack : jxrnfzawmzhic__________NYSLXGHD
Iteration  46 : Push s in stack 1. New stack : jxrnfzawmzhics_________NYSLXGHD
Iteration  47 : Push X in stack 2. New stack : jxrnfzawmzhics________XNYSLXGHD
Iteration  48 : Push q in stack 1. New stack : jxrnfzawmzhicsq_______XNYSLXGHD
Iteration  49 : Push v in stack 1. New stack : jxrnfzawmzhicsqv______XNYSLXGHD
Iteration  50 : Push a in stack 1. New stack : jxrnfzawmzhicsqva_____XNYSLXGHD
Iteration  51 : Push l in stack 1. New stack : jxrnfzawmzhicsqval____XNYSLXGHD
Iteration  52 : Push l in stack 1. New stack : jxrnfzawmzhicsqvall___XNYSLXGHD
Iteration  53 : Push h in stack 1. New stack : jxrnfzawmzhicsqvallh__XNYSLXGHD
Iteration  54 : Pop    in stack 1. New stack : jxrnfzawmzhicsqvall___XNYSLXGHD
Iteration  55 : Push k in stack 1. New stack : jxrnfzawmzhicsqvallk__XNYSLXGHD
Iteration  56 : Push x in stack 1. New stack : jxrnfzawmzhicsqvallkx_XNYSLXGHD
Iteration  57 : Push q in stack 1. New stack : jxrnfzawmzhicsqvallkxq_XNYSLXGHD
Iteration  58 : Push F in stack 2. New stack : jxrnfzawmzhicsqvallkxqFXNYSLXGHD
Iteration  59 : Pop    in stack 2. New stack : jxrnfzawmzhicsqvallkxq_XNYSLXGHD
Iteration  60 : Push A in stack 2. New stack : jxrnfzawmzhicsqvallkxqAXNYSLXGHD
Iteration  61 : Pop    in stack 1. New stack : jxrnfzawmzhicsqvallkx_AXNYSLXGHD
Iteration  62 : Push z in stack 1. New stack : jxrnfzawmzhicsqvallkxzAXNYSLXGHD
Iteration  63 : Push v in stack 1. New stack : Error: Overflow in stack.

Enter strategy -- 0 (odd-even) or 1 (colliding) : 1
Iteration   1 : Push C in stack 2. New stack : _____________________________C
Iteration   2 : Push z in stack 1. New stack : z____________________________C
Iteration   3 : Push n in stack 1. New stack : zn___________________________C
Iteration   4 : Push h in stack 1. New stack : znh__________________________C
Iteration   5 : Push k in stack 1. New stack : znhk_________________________C
Iteration   6 : Push q in stack 1. New stack : znhkq________________________C
Iteration   7 : Push e in stack 1. New stack : znhkqe_______________________C
Iteration   8 : Push a in stack 1. New stack : znhkqea______________________C
Iteration   9 : Push t in stack 1. New stack : znhkqeat_____________________C
Iteration  10 : Push z in stack 1. New stack : znhkqeatz____________________C
Iteration  11 : Push f in stack 1. New stack : znhkqeatzf___________________C
Iteration  12 : Pop    in stack 1. New stack : znhkqeatz____________________C
Iteration  13 : Push x in stack 1. New stack : znhkqeatzx___________________C
Iteration  14 : Push Z in stack 2. New stack : znhkqeatzx__________________ZC
Iteration  15 : Pop    in stack 2. New stack : znhkqeatzx___________________C
Iteration  16 : Push e in stack 1. New stack : znhkqeatzxe__________________C
Iteration  17 : Push q in stack 1. New stack : znhkqeatzxeq_________________C
Iteration  18 : Push t in stack 1. New stack : znhkqeatzxeqt________________C
Iteration  19 : Push v in stack 1. New stack : znhkqeatzxeqtv_______________C
Iteration  20 : Push r in stack 1. New stack : znhkqeatzxeqtvr______________C
Iteration  21 : Push Q in stack 2. New stack : znhkqeatzxeqtvr_____________QC
Iteration  22 : Push v in stack 1. New stack : znhkqeatzxeqtvrv____________QC
Iteration  23 : Pop    in stack 1. New stack : znhkqeatzxeqtvr_____________QC
Iteration  24 : Pop    in stack 2. New stack : znhkqeatzxeqtvr_____________C
Iteration  25 : Pop    in stack 1. New stack : znhkqeatzxeqtv______________C
Iteration  26 : Pop    in stack 2. New stack : znhkqeatzxeqtv_______________
Iteration  27 : Push v in stack 1. New stack : znhkqeatzxeqtvv______________
Iteration  28 : Push k in stack 1. New stack : znhkqeatzxeqtvvk_____________
Iteration  29 : Push i in stack 1. New stack : znhkqeatzxeqtvvki____________
Iteration  30 : Pop    in stack 1. New stack : znhkqeatzxeqtvvk_____________
Iteration  31 : Pop    in stack 2. New stack : Error: Underflow in Stack 2.
```