```
runner% cat -n listss.c
     1  /* listss: test recursive printing of linked lists */
     2  #include <stdio.h>
     3  #include <stdlib.h>
     4
     5  struct snode {           /* self-referential struct */
     6     char c;               /* char field */
     7     struct snode *next;   /* pointer to same kind of struct */
     8  };
     9
    10  char pop(void);                         /* pop an item. */
    11  void push(char);                        /* push an item. */
    12  int empty(void);                        /* check if empty */
    13  int full(void);                         /* check if full (never is) */
    14  void printstack(struct snode *);        /* print in top-down order */
    15  void printstackrecurs(struct snode *);  /* print recursively top town */
    16  void printstackreverse(struct snode *); /* print recursively bot up */
    17
    18  struct snode *s = NULL;  /* global variable for simplicity */
    19
    20  void main(void)
    21  {
    22     char ch;
    23     struct snode *t;
    24     while ((ch = getchar()) != '\n' && !full()) { /* read a line */
    25        push(ch);
    26     }
    27     printstack(s); putchar('\n');
    28     printstackrecurs(s); putchar('\n');
    29     printstackreverse(s); putchar('\n');
    30     for (t = s; t != NULL; t = t -> next)
    31        putchar(t -> c);
    32     putchar('\n');
    33  }
    34
    35  /* pop: pop the linked list stack (remove from front) */
    36  char pop(void)
    37  {
    38     char ch;
    39     struct snode *ssave;
    40     if (empty()) {
    41        fprintf(stderr, "Stack underflow\n");
    42        exit(1);
    43     }
    44     ch = s -> c;
    45     ssave = s;
    46     s = s -> next;
    47     free(ssave);
    48     return ch;
    49  }
    50
    51  /* push: push onto the linked list stack (add at front) */
    52  void push(char ch)
    53  {
    54     struct snode *t = (struct snode *)
    55        malloc(sizeof(struct snode));
    56     if (t == NULL) {
    57        fprintf(stderr, "Allocation failed\n");
    58        exit(1);
    59     }
    60     t -> next = s;
    61     t -> c = ch;
    62     s = t;
    63  }
    64
    65  /* empty: NULL pointer indicates empty stack */
    66  int empty(void)
    67  {
    68     return s == NULL;
    69  }
    70
    71  /* full: stack is only full if storage gives out */
    72  int full(void)
    73  {
    74     return 0;
    75  }
    76
    77  /* printstack: chase down stack, printing as you go */
    78  void printstack(struct snode *s)
    79  {
    80     while (s != NULL) {  /* standard while loop */
    81        putchar(s -> c);
    82        s = s -> next;
    83     }
    84  }
    85
    86  /* printstackrecurs: chase down stack and print, but recursively */
    87  void printstackrecurs(struct snode *s)
    88  {
    89     if (s != NULL) {  /* Note: no loop, just an if */
    90        putchar(s -> c);
    91        printstackrecurs(s -> next);
    92     }
    93  }
    94
    95  /* printstackreverse: chase down stack and chase again recursively, */
    96  /*    before printing */
    97  void printstackreverse(struct snode *s)
    98  {
    99     if (s != NULL) {  /* Note: no loop, just an if */
   100        printstackreverse(s -> next);
   101        putchar(s -> c);
   102     }
   103  }
runner% lint -m -u listss.c

function returns value which is always ignored
    fprintf             putchar
runner% cc -o listss listss.c
runner%% listss
Now is the time for all good men to come to the aid of their party.
.ytrap rieht fo dia eht ot emoc ot nem doog lla rof emit eht si woN
.ytrap rieht fo dia eht ot emoc ot nem doog lla rof emit eht si woN
Now is the time for all good men to come to the aid of their party.
.ytrap rieht fo dia eht ot emoc ot nem doog lla rof emit eht si woN
runner%
```