Listing 1: `shd_stack.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef int item_t;
5
6  typedef struct { item_t  *base;
7                   int      size;
8                   int  max_size;
9                   item_t  *copy;
10                  int copy_size; }          stack_t;
11
12 stack_t *create_stack(int size)
13 {   stack_t *st;
14     st = (stack_t *) malloc( sizeof(stack_t) );
15     st->base = (item_t *) malloc( size * sizeof(item_t) );
16     st->max_size = size;
17     st->size = 0; st->copy = NULL; st->copy_size = 0;
18     return( st );
19 }
20
21 int stack_empty(stack_t *st)
22 {   return( st->size == 0);
23 }
24
25 void push( item_t x, stack_t *st)
26 {   *(st->base + st->size) = x;
27     st->size += 1;
28     if ( st->copy != NULL || st->size >= 0.75*st->max_size )
29     {  /* have to continue or start copying */
30        int additional_copies = 4;
31        if( st->copy == NULL ) /* start copying: allocate space */
32        {  st->copy =
33           (item_t *) malloc( 2 * st->max_size * sizeof(item_t) );
34        }
35        /* continue copying: at most 4 items per push operation */
36        while( additional_copies > 0 &&
37               st->copy_size < st->size )
38        {  *(st->copy + st->copy_size) =
39                                    *(st->base + st->copy_size);
40           st->copy_size += 1; additional_copies -= 1;
41        }
42        if( st->copy_size == st->size) /* copy complete */
43        {  free( st->base );
44           st->base = st-> copy;
```

```
45              st->max_size *= 2;
46              st->copy = NULL;
47              st->copy_size = 0;
48          }
49      }
50  }
51
52  item_t pop(stack_t *st)
53  {   item_t tmp_item;
54      st->size -= 1;
55      tmp_item = *(st->base + st->size);
56      if( st->copy_size == st->size) /* copy complete */
57      {   free( st->base );
58          st->base = st-> copy;
59          st->max_size *= 2;
60          st->copy = NULL;
61          st->copy_size = 0;
62      }
63      return( tmp_item );
64  }
65
66  item_t top_element(stack_t *st)
67  {   return( *(st->base + st->size - 1) );
68  }
69
70  void remove_stack(stack_t *st)
71  {   free( st->base );
72      if( st->copy != NULL )
73          free( st->copy );
74      free( st );
75  }
76
77  void list_stack(stack_t *st)
78  { int i;
79    printf("The items currently on the stack are, from the top\n");
80    for( i= st->size -1; i>=0; i -- )
81       printf("%d ", *(st->base + i ) );
82    if( st->copy != NULL )
83    { printf("A copy is being constructed, the items on the copy are\n");
84      for( i= st->copy_size -1; i>=0; i -- )
85         printf("%d ", *(st->copy + i ) );
86      printf("\n");
87    }
88    else
89      printf("no copy exists now\n");
```

```
90  }
91
92  int main()
93  {   stack_t *st;
94      char nextop;
95      st = create_stack(3);
96      printf("Made Stack\n");
97      while( (nextop = getchar())!= 'q' )
98      { if( nextop == 'a' )
99        { int addkey;
100         scanf(" %d", &addkey);
101         push( addkey, st );
102         printf("pushed item %d on stack\n", addkey);
103       }
104       if( nextop == 'p' )
105       {  printf("popped item %d from stack\n", pop(st) );
106       }
107       if( nextop == '?' )
108       { if( stack_empty(st) )
109       printf("The stack is empty\n");
110         else
111         {  printf("The top item on the stack is %d,", top_element(st));
112            printf("the stack size is %d\n", st->size );
113            list_stack(st);
114         }
115       }
116     }
117     remove_stack(st);
118     printf(" Removed stack.\n");
119     return(0);
120 }
```