You are not allowed to use loop constructs like while, for..etc, and you can only use the following ADT functions on Stack S:
isEmpty(S)
push(S)
pop(S)

Solution:
The idea of the solution is to hold all values in Function Call Stack until the stack becomes empty. When the stack becomes empty, insert all held items one by one at the bottom of the stack.

For example, let the input stack be

```
    1  <-- top
    2
    3
    4
```

```
First 4 is inserted at the bottom.
    4 <-- top

Then 3 is inserted at the bottom
    4 <-- top
    3

Then 2 is inserted at the bottom
    4 <-- top
    3
    2

Then 1 is inserted at the bottom
    4 <-- top
    3
    2
    1
```

So we need a function that inserts at the bottom of a stack using the above given basic stack function. **//Below is a recursive function that inserts an element at the bottom of a stack.**

```
void insertAtBottom(struct sNode** top_ref, int item)
{
   int temp;
   if(isEmpty(*top_ref))
   {
      push(top_ref, item);
   }
   else
   {

     /* Hold all items in Function Call Stack until we reach end of
       the stack. When the stack becomes empty, the isEmpty(*top_ref)
       becomes true, the above if part is executed and the item is
       inserted at the bottom */
     temp = pop(top_ref);
     insertAtBottom(top_ref, item);

     /* Once the item is inserted at the bottom, push all the
         items held in Function Call Stack */
     push(top_ref, temp);
```

```
    }
}
```

//Below is the function that reverses the given stack using insertAtBottom()

```
void reverse(struct sNode** top_ref)
{
  int temp;
  if(!isEmpty(*top_ref))
  {

    /* Hold all items in Function Call Stack until we reach end of
     the stack */
    temp = pop(top_ref);
    reverse(top_ref);

    /* Insert all the items (held in Function Call Stack) one by one
       from the bottom to top. Every item is inserted at the bottom */
    insertAtBottom(top_ref, temp);
  }
}
```

//Below is a complete running program for testing above functions.

```
#include<stdio.h>
#include<stdlib.h>
#define bool int

/* structure of a stack node */
struct sNode
{
   char data;
   struct sNode *next;
};

/* Function Prototypes */
void push(struct sNode** top_ref, int new_data);
int pop(struct sNode** top_ref);
bool isEmpty(struct sNode* top);
void print(struct sNode* top);

/* Driveer program to test above functions */
int main()
{
  struct sNode *s = NULL;
  push(&s, 4);
  push(&s, 3);
  push(&s, 2);
  push(&s, 1);

  printf("\n Original Stack ");
  print(s);
  reverse(&s);
  printf("\n Reversed Stack ");
  print(s);
  getchar();
}

/* Function to check if the stack is empty */
bool isEmpty(struct sNode* top)
{
  return (top == NULL)? 1 : 0;
}

/* Function to push an item to stack*/
void push(struct sNode** top_ref, int new_data)
{
  /* allocate node */
  struct sNode* new_node =
            (struct sNode*) malloc(sizeof(struct sNode));

  if(new_node == NULL)
  {
     printf("Stack overflow \n");
     getchar();
     exit(0);
  }

  /* put in the data  */
  new_node->data  = new_data;

  /* link the old list off the new node */
  new_node->next = (*top_ref);

  /* move the head to point to the new node */
  (*top_ref)    = new_node;
}

/* Function to pop an item from stack*/
int pop(struct sNode** top_ref)
{
  char res;
  struct sNode *top;

  /*If stack is empty then error */
  if(*top_ref == NULL)
  {
     printf("Stack overflow \n");
     getchar();
     exit(0);
  }
  else
  {
     top = *top_ref;
     res = top->data;
     *top_ref = top->next;
     free(top);
     return res;
  }
}

/* Functrion to pront a linked list */
void print(struct sNode* top)
{
  printf("\n");
  while(top != NULL)
  {
    printf(" %d ", top->data);
    top =  top->next;
  }
}
```