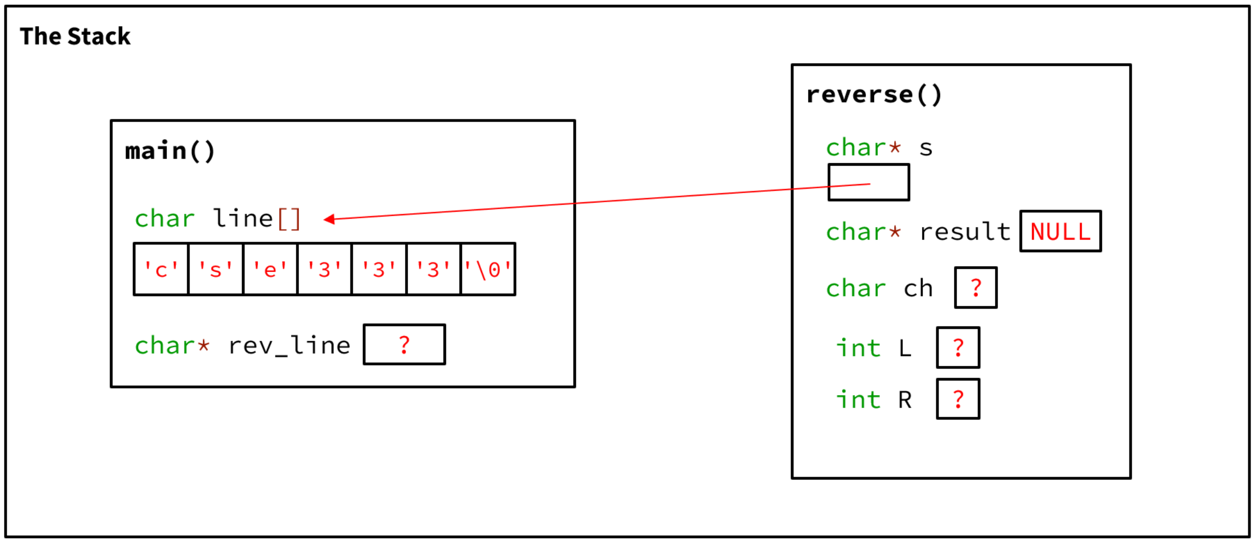
CSE 333 22au – Section 2: Structs and Debugging SOLUTION

Exercise 1

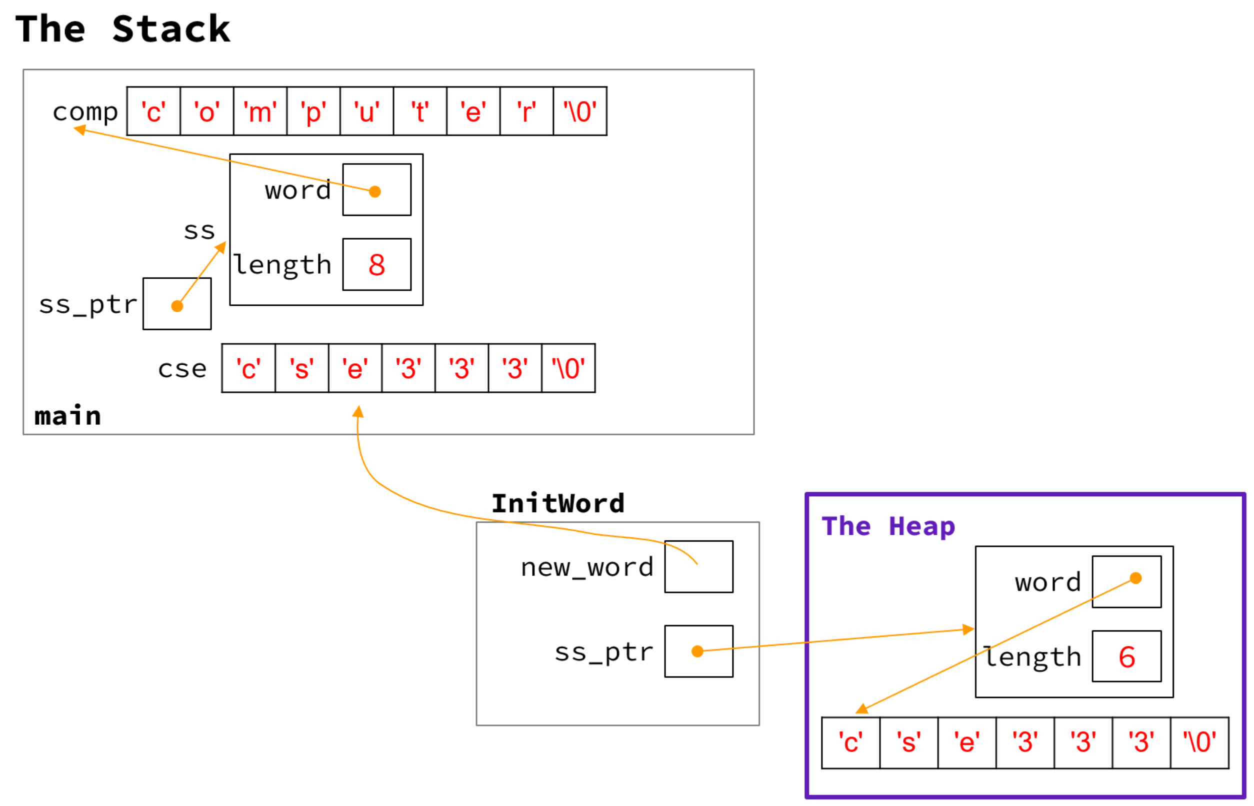
Draw a memory diagram for the execution of the code above up to the call to `strncpy()` in `reverse()`. Make sure to distinguish between local variables on the Stack and Heap-allocated memory.



Exercise 2 and 3

```
/*
 * Ask user for a word and print it forwards and backwards.
 * CSE 333 demo (for debugging).   HP
 */

#define MAX_STR 100   /* length of longest input string */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Return a new string with the contents of s backwards */
char * reverse(char * s) {
  char * result = NULL;             /* the reversed string */
  int L, R;
  char ch;

  /* copy original string then reverse and return the copy */
  int strsize = strlen(s)+1;
  /* EX2 Fix: Allocate space for reversed string*/
  result = (char *)malloc(strsize);
  /* EX3 Fix: Check for malloc failure */
  if (result == NULL) {
    exit(EXIT_FAILURE);
  }
  strncpy(result, s, strsize);

  L = 0;
  R = strlen(result) - 1;
  while (L < R) {
    ch = result[L];
    result[L] = result[R];
    result[R] = ch;
    L++; R--;
  }

  return result;
}

/* Ask the user for a string, then print it forwards and backwards.
*/
int main() {
  char line[MAX_STR];     /* original input line */
  char * rev_line;        /* backwards copy from reverse function */

  printf("Please enter a string: ");
  fgets(line, MAX_STR, stdin);
  line[strlen(line)-1] = '\0';
  rev_line = reverse(line);
  printf("The original string was:   >%s<\n", line);
  printf("Backwards, that string is: >%s<\n", rev_line);
  printf("Thank you for trying our program.\n");
  /* EX2 Fix: Free the reversed string preventing memory leak */
  free(rev_line);
  return EXIT_SUCCESS;
}
```

Exercise 4


The Stack
comp
'c' 'o' 'm' 'p' 'u' 't' 'e' 'r' '\0'
word
ss
length
8
ss_ptr
cse
'c' 's' 'e' '3' '3' '3' '\0'
main
InitWord
new_word
ss_ptr
The Heap
word
length
6
'c' 's' 'e' '3' '3' '3' '\0'

Exercise 5 (Bonus)

Also added malloc checks.

```
void InitWord(char* word, SimpleString** dest) {
   *dest = (SimpleString*) malloc(sizeof(SimpleString));
   if (*dest == NULL) {
      exit(EXIT_FAILURE);
   }
   (*dest)->length = strlen(word);
   (*dest)->word = (char*) malloc(sizeof(char)*((*dest)->length + 1));
   if ((*dest)->word == NULL) {
      exit(EXIT_FAILURE);
   }
   strncpy((*dest)->word, word, (*dest)->length + 1);
}
```