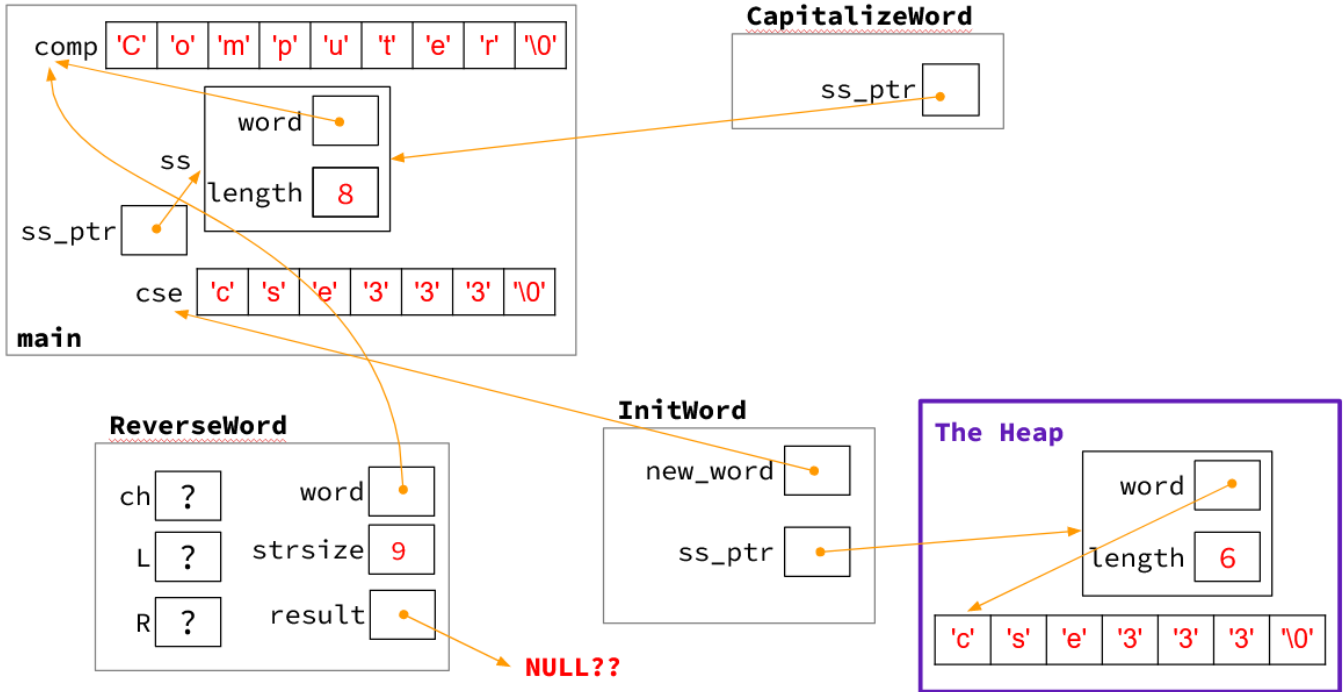


CSE 333 – Section 2: Structs and Debugging Solutions

Exercise 1

Below is the state of memory up to the call to `strcpy()` in `ReverseWord()`. Functions whose stack frames have been popped are shown for completion.

The Stack



Exercise 2

Note: To see the full solution with proper use of function declarations on the course website.

```
/* Fixed code for CSE 333 Section 2
 * 1. Draw a memory diagram for the execution to identify errors.
 * 2. Use gdb and valgrind to identify sources of runtime, logical,
 *    and memory errors.
 * 3. Clean up the code style.
 */

#include <string.h> // strncpy, strlen
#include <stdio.h> // printf
#include <stdlib.h> // malloc, NULL, free, EXIT_FAILURE, EXIT_SUCCESS
#include <ctype.h> // toupper

// A SimpleString stores a C-string and its current length
typedef struct simplestring_st {
    char* word;
    int length;
} SimpleString;

// Capitalize the first letter in the word
void CapitalizeWord(SimpleString* ss_ptr);

/*
 * FIXED: Returning a pointer as an output parameter requires a double
 pointer
 */
// Allocate a new SimpleString on the heap initialized with word
// and return pointer to the new SimpleString in dest
void InitWord(char* word, SimpleString** dest);

// Return a new string with the characters in word in reverse order.
char* ReverseWord(char* word);

int main(int argc, char* argv[]) {
    char comp[] = "computer";
    SimpleString ss = {comp, strlen(comp)};
    SimpleString* ss_ptr = &ss;

    // expecting "1. computer, 8"
    printf("1. %s, %d\n", ss_ptr->word, ss_ptr->length);

    char cse[] = "cse333";
    InitWord(cse, &ss_ptr);
    // expecting "2. cse333, 6"
    printf("2. %s, %d\n", ss_ptr->word, ss_ptr->length);

    CapitalizeWord(ss_ptr);
    // expecting "3. Cse333, 6"
    printf("3. %s, %d\n", ss_ptr->word, ss_ptr->length);
}
```

```

char* reversed = ReverseWord(ss_ptr->word);
/*
 * FIXED: Use free to fix memory leaks from InitWord
 * (Use valgrind to find this leak)
 */
free(ss_ptr->word);
free(ss_ptr);
InitWord(reversed, &ss_ptr);
// expecting "4. 333esC, 6"
printf("4. %s, %d\n", ss_ptr->word, ss_ptr->length);

/*
 * FIXED: Use free to fix memory leaks from heap-allocated reversed
string
 * (Use valgrind to find this leak)
 */
free(reversed);
free(ss_ptr->word);
free(ss_ptr);
return EXIT_SUCCESS;
}

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);
}

void CapitalizeWord(SimpleString* ss_ptr) {
    ss_ptr->word[0] = toupper(ss_ptr->word[0]);
}

char* ReverseWord(char* word) {
    char* result = NULL; // the reversed string
    int L, R;
    char ch;

    // copy original string then reverse and return the copy
    int strsize = strlen(word) + 1;
    /*
     * FIXED: Return string is heap-allocated and responsibility to free
passed to
     * the caller

```

```

*/
result = (char*) malloc(strsize);
if (result == NULL) {
    exit(EXIT_FAILURE);
}
strncpy(result, word, strsize);

L = 0;
/*
 * FIXED: Decrement length of result by 1 to avoid copying null
terminator
 */
R = strlen(result) - 1;
while (L < R) {
    ch = result[L];
    result[L] = result[R];
    result[R] = ch;
    L++;
    R--;
}

return result;
}

```

Exercise 3

Style Issues

1. Should error check any call to malloc()

```

if (rev == NULL) {
    return NULL;
}
if (rev->word == NULL) {
    return NULL;
}

```