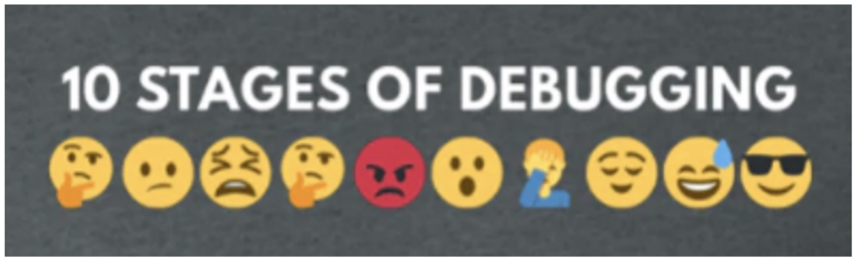
CSE 333 Section 2

Structs and Debugging



UNIVERSITY *of* WASHINGTON

Checking In & Logistics

- Exercise 4:
  - Due **Friday @ 10:00am (4/8)**
- Exercise 5:
  - Due **Monday @ 10:00am (4/11)**
- Homework 1:
  - Due **Thursday @ 11:00pm (4/14)**
  - Start Early!

Any questions, comments, or concerns?

- Exercises going ok?
- Lectures making sense?

Structs and Typedef Review

Defining Structs

- To define a struct, we use the `struct` statement, which typically has a name (a tag) and must have one or more data members
    - This defines a new data type!

```c
struct simplestring_st {
  char* word;
  int   length;
};
struct simplestring_st my_word;
```

Typedef

- The C Programming language provides the keyword `typedef`, which defines an alias (alternate name) for an existing data type
  - This can be used in combination with a `struct` statement

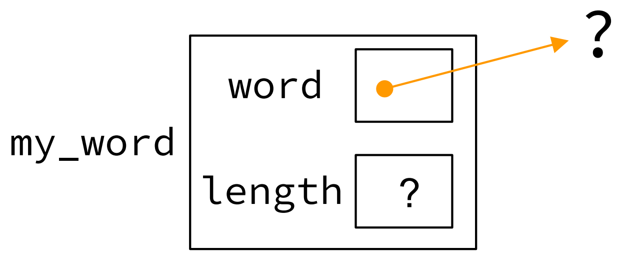
```c
struct simplestring_st {
  char* word;
  int   length;
};
typedef struct simplestring_st SimpleString;
SimpleString my_word;
```

```c
typedef struct simplestring_st {
  char* word;
  int   length;
} SimpleString;
SimpleString my_word;
```

Structs and Memory Diagrams

- `struct` instance is a box, with individual boxes for fields inside of it, labelled with field names
  - Even though we know that field ordering is guaranteed, we can be loose with where we place the fields in our diagram

```
typedef struct simplestring_st {
  char* word;
  int   length;
} SimpleString;
SimpleString my_word;
```

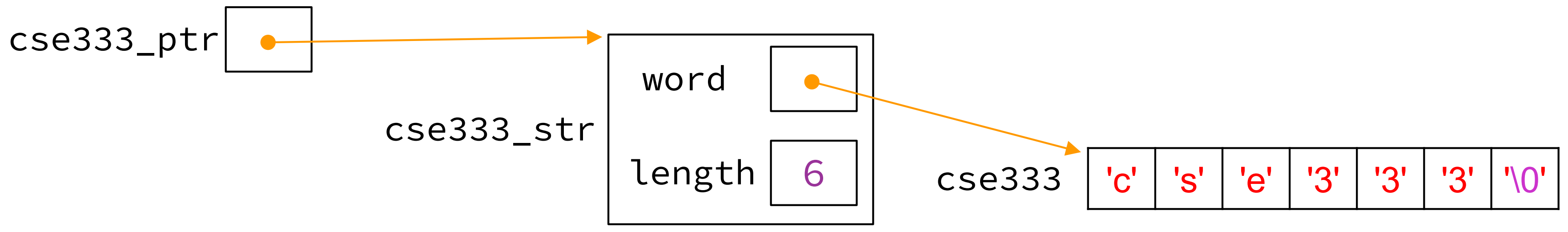


Structs and Pointers

- "`.`" to access field from `struct` instance
- "`->`" to access field from `struct` pointer

```
typedef struct simplestring_st {
  char* word;
  int   length;
} SimpleString;
```

```
char cse333[] = "cse333";
SimpleString  cse333_ss;
SimpleString* cse333_ptr = &cse333_ss;

cse333_count.word = cse333_ss;
cse333_ptr->length = strlen(cse333);
```



Passing Structs as Parameters

- Assignment copies over all of the field values
  - Unlike reference copying in Java

- Structs are *pass-by-copy* (as arguments and return values)
  - Can imitate pass-by-reference by passing pointer to struct instance instead

Trying to Run `simplestring.c`

We have a program *`simplestring.c`* that uses the struct `SimpleString` which keeps track of both a C-string and the length of the C-string.

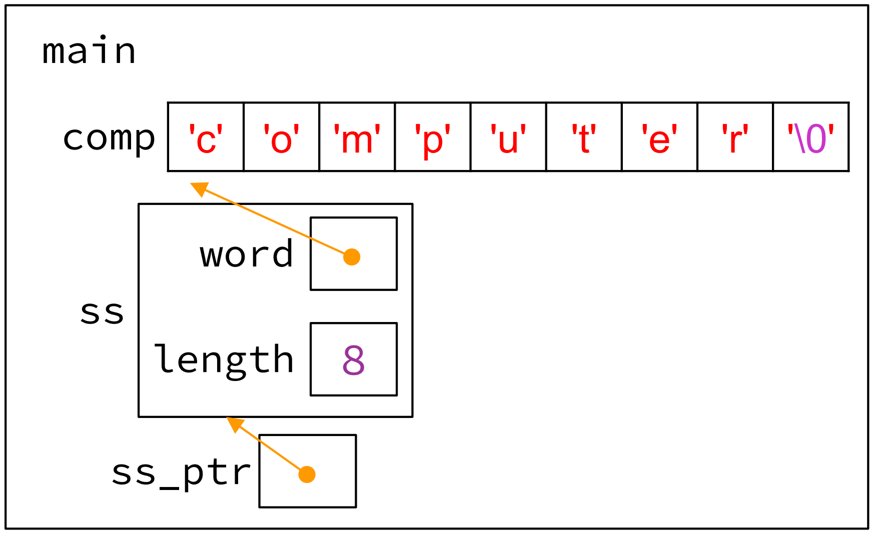
But it has a few problems… let's take a look!

Exercise 1

Note: boxes with a function name above are local variables on the stack

Complete the Memory Diagram

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

  printf("1. %s, %d\n", ss_ptr->word,
         ss_ptr->length);
  ...
}
```

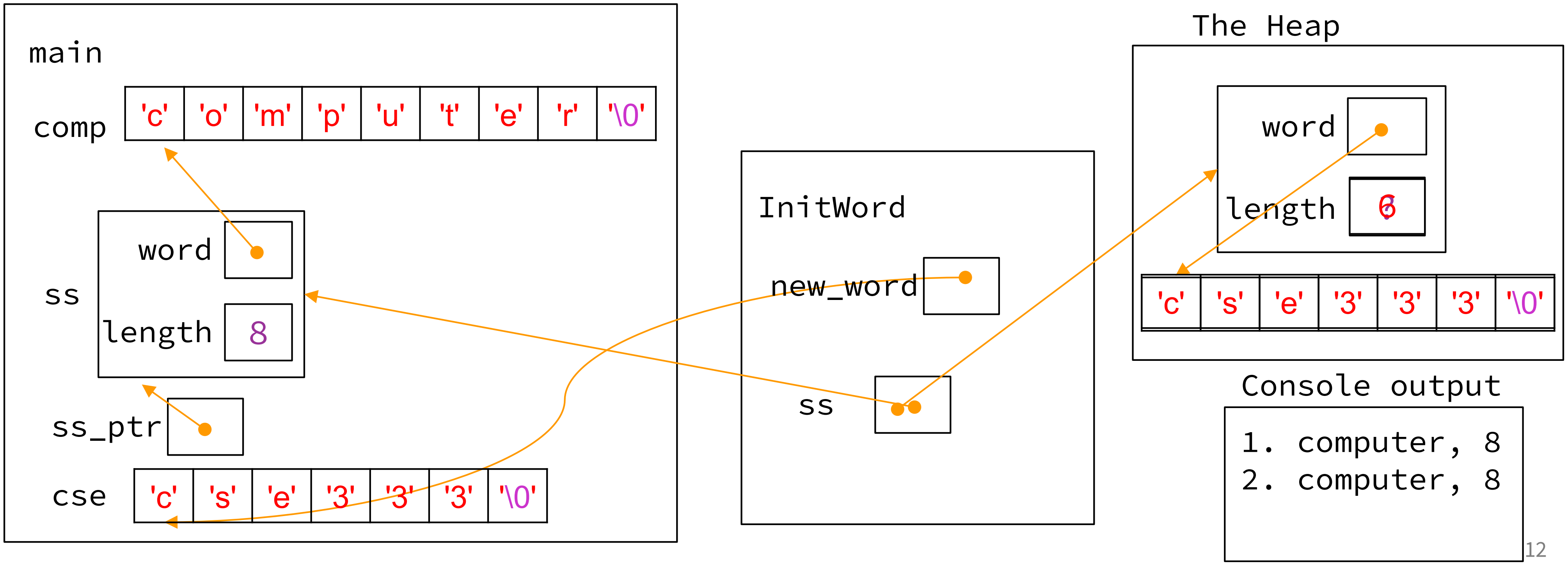


Console output

```
1. computer, 8
```

```
    // continued main code
    char cse[] = "cse333";
    InitWord(cse333, ss_ptr);
    printf("2. %s, %d\n", ss_ptr->word,
           ss_ptr->length);
    ...
}
```

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

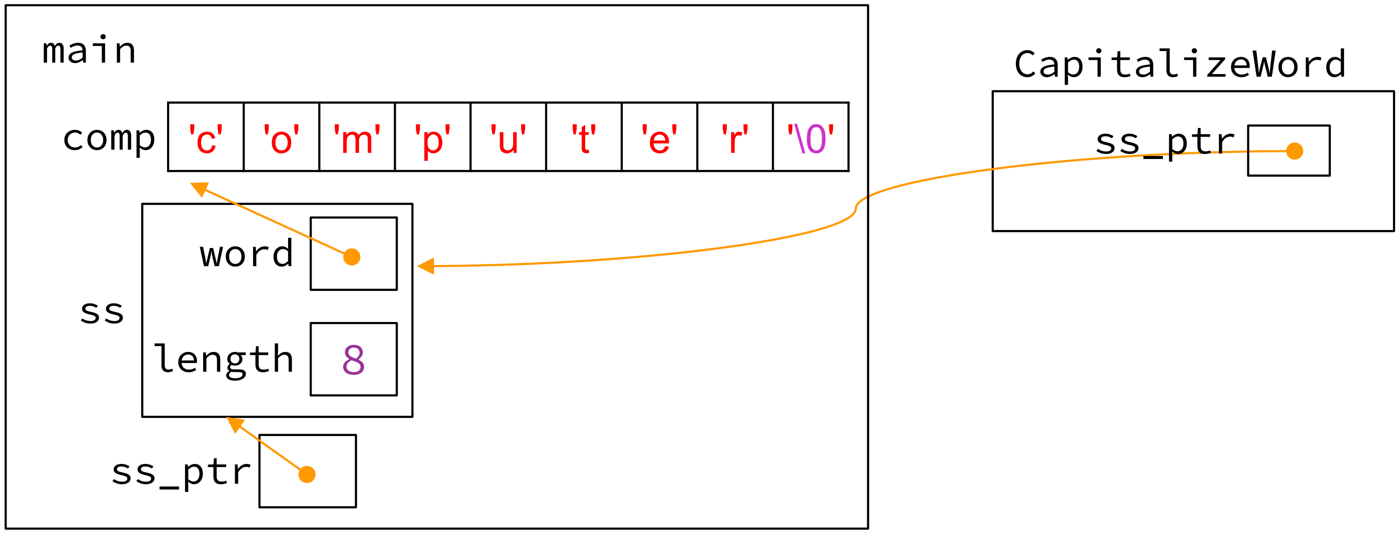


main
comp
'c' 'o' 'm' 'p' 'u' 't' 'e' 'r' '\0'
ss
word
length
8
ss_ptr
cse
'c' 's' 'e' '3' '3' '3' '\0'
InitWord
new_word
ss
The Heap
word
length
6
'c' 's' 'e' '3' '3' '3' '\0'
Console output

```
1. computer, 8
2. computer, 8
```

```
  // continued main code
  CapitalizeWord(ss_ptr);
  printf("3. %s, %d\n",
         ss_ptr->word,
         ss_ptr->length);
  ...
}
```

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

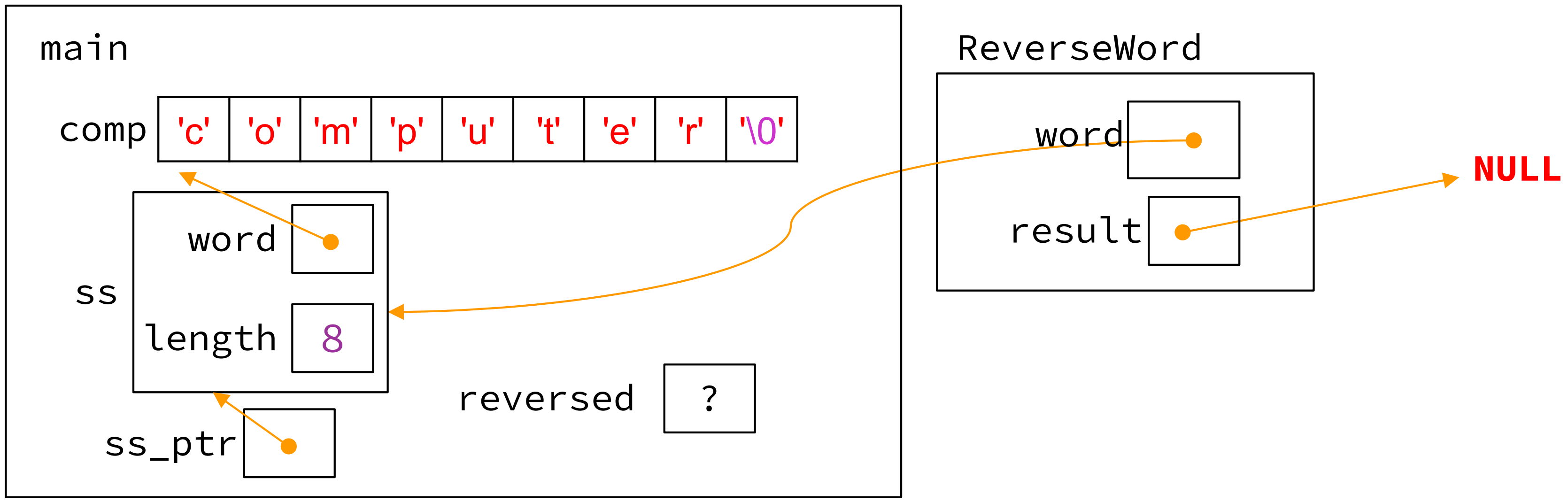


Console output

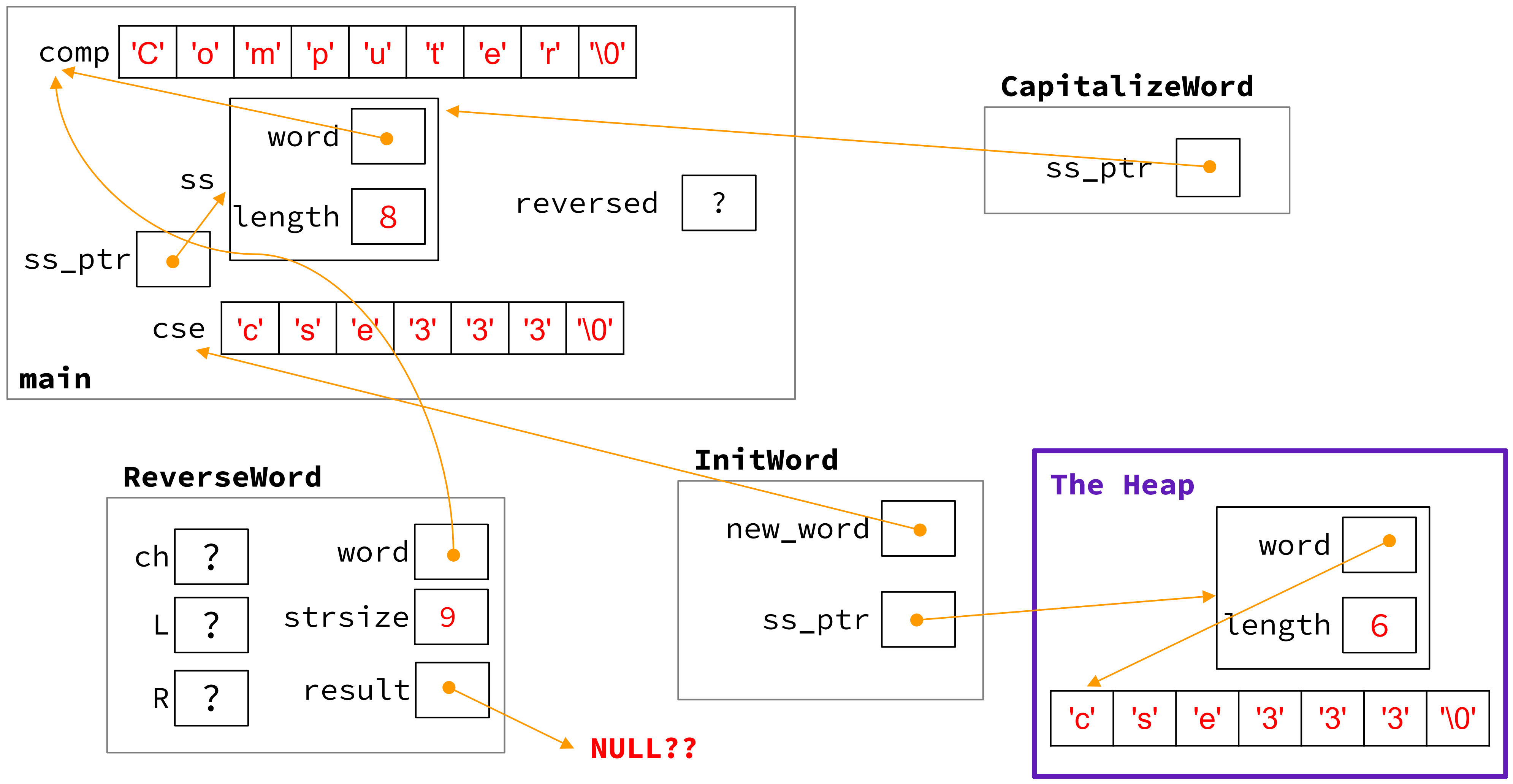
```
1. computer, 8
2. computer, 8
3. Computer, 8
```

```
// ...continued main code
char* reversed = ReverseWord(ss_ptr->word);

...
```

```
char* ReverseWord(char* word) {
  char* result = NULL;
  int strsize = strlen(word) + 1;

  strncpy(result, word, strsize);
  ...
}
```



The Stack


comp
'C' 'o' 'm' 'p' 'u' 't' 'e' 'r' '\0'
ss
word
length
8
reversed
?
ss_ptr
cse
'c' 's' 'e' '3' '3' '3' '\0'
main
CapitalizeWord
ss_ptr
ReverseWord
ch
?
L
?
R
?
word
strsize
9
result
NULL??
InitWord
new_word
ss_ptr
The Heap
word
length
6
'c' 's' 'e' '3' '3' '3' '\0'

Debugging Tools

Debugging

- ✨ **Debugging is a skill that you will need throughout your career!** ✨
- The 333 projects are big with lots of potential for bugs
  - Learning to use the debugging tools will make your life a lot easier
  - Course staff will help you learn the tools in office hours, too
- Debugging tool output can be scary at first, but extremely useful once you know how to parse it

Debugging

Many debugging strategies exist but here's a simple 5 step process!

1. **Observation**: Something is wrong with your program!
2. **Hypothesis**: What do you think is going wrong?
3. **Measurement**: Use debuggers and other tools to verify the problem
4. **Analyze**: Identify and implement a fix to the problem.
5. Repeat steps 1-4 until *bug free*!

333 Debugging Options

- gdb (GNU Debugger) is a general-purpose debugging tool
  - Stops at breakpoints and program crashes
  - Lots of helpful features for tracing code, checking current expression values, and examining memory
- `valgrind` specifically check for memory errors
  - Great for catching non-crashing odd behavior (*e.g.*, using uninitialized values, memory leaks on the heap)
  - If your code uses `malloc`, should use `--leak-check=full` option

Tracing Code in gdb

- Reference Card: https://courses.cs.washington.edu/courses/cse333/22sp/resources/gdb-refcard.pdf

- Setting breakpoints:
  - `break <filename:line#>`
- Advancing
  - `step` – into functions
  - `next` – over functions
  - `continue` – to next break
- Reading Values
  - `print` – evaluate expression once
  - `display` – keep evaluating expression
- Examining memory
  - `x` – dereference provided address

Common Errors

```
Hello World!
Segmentation fault (core dumped)
```

- **Misusing Functions**: Read documentation (online, through man pages, or the `.h` files for your homework) for function parameters and function purpose
  - Oftentimes, this leads to unexpected results!

- **Segmentation Fault**: Dereferencing an uninitialized pointer, `NULL`, a previously-freed pointer, or many other things.
  - GDB automatically halts execution when `SIGSEGV` is received, useful for debugging

- **Memory “Errors”**: Many possible errors, commonly use of uninitialized memory or “memory leaks” (data allocated on heap that does not get free’d).
  - Use `valgrind` to help catch memory errors!

Exercise 2

Fix 1: Segfault

- Tool help: `run` in gdb to find segfault, `man` for `strncpy`

- Old version:

```
result = NULL;
strncpy(result, word, strsize);
```

- New version:

```
result = (char*) malloc(strsize);
strncpy(result, word, strsize);
```

Fix 2: Doesn't initialize word

- Tool help: Stepping through with gdb

Old version:

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

New version:

```
void InitWord(char* word, SimpleString** dest) {
  *dest = (SimpleString*)
       malloc(sizeof(SimpleString));

  (*dest)->length = strlen(word);
  (*dest)->word = (char*)
       malloc(sizeof(char) * ((*dest)->length + 1));

  strncpy((*dest)->word, word,
           (*dest)->length + 1);
}
```

Fix 3: Doesn't reverse string

- Tool help: run in gdb, break on `ReverseWord`, step through code, `print /s word` at end of function (prints as string)
- Old version:

```
char ch;
int L = 0, R = strlen(result);
```

- New version:

```
char ch;
int L = 0, R = strlen(result) - 1;
```

Fix 4: Memory leaks

- Tool help: run under `valgrind`, identify un-freed allocation line numbers

- Old version:

```
char* ReverseWord(char* word) { ...
return result; }
```

- New version:

```
char* ReverseWord(char* word) { ...
return result; }
At end of main:   free(ss_ptr->word); or free(ss.word);
```

Exercise 3

Style Fixes

- Tool help: None? Lecture slides! Google C++ Style Guide!
- `malloc` error checking:

```
result = (char*) malloc(strsize);
if (result == NULL) {
  // sample error checking. Read the spec on the requirements
  // for handling malloc!
  exit(EXIT_FAILURE);
}
```