Homework 1

1. The following program crashes because when we allocate memory for d[len+1] it takes up memory in the stack segment as another stack frame is allocated to contain local variables of strdup(). However, when the function strdup() returns a value, this frame is popped from the stack. Thus when we try to put the value of strdup(argv[i]) to the terminal in the main function it gives random values.
We can remedy this by setting d as static char d[100]. This way the char d is allocated in the data segment and stays after the function strdup() is popped from stack.

2.
Text segment:

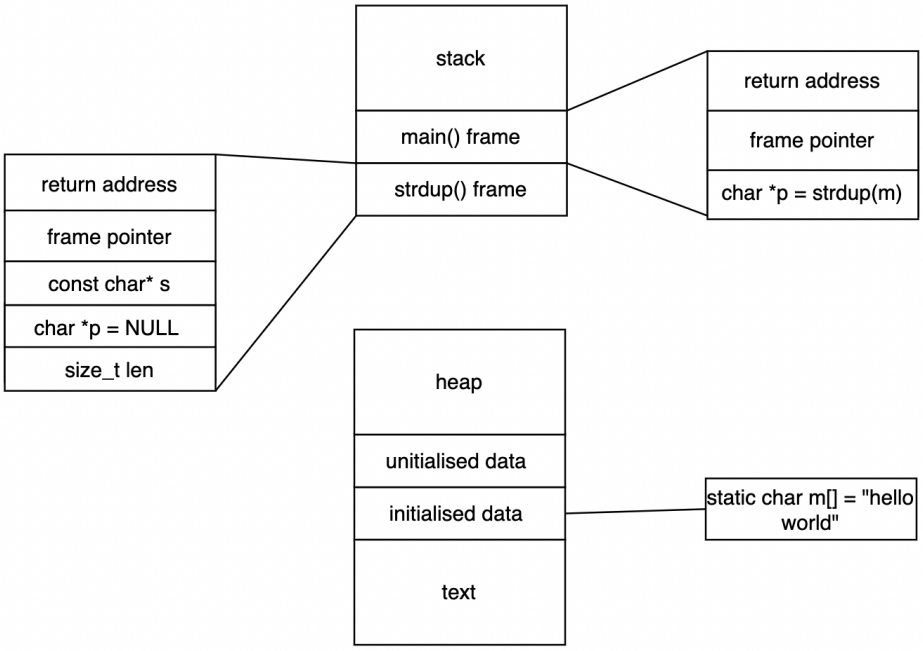
```
        .section __TEXT,__text,regular,pure_instructions
        .build_version macos, 11, 0        sdk_version 12, 1
        .globl    _strdup                  ## -- Begin function strdup
        .p2align 4, 0x90
_strdup:                                ## @strdup
        .cfi_startproc
## %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq    $32, %rsp
        movq    %rdi, -8(%rbp)
        movq    $0, -16(%rbp)
        cmpq    $0, -8(%rbp)
        je      LBB0_4
## %bb.1:
        movq    -8(%rbp), %rdi
        callq   _strlen
```

```
	movq	%rax, -24(%rbp)
	movq	-24(%rbp), %rdi
	addq	$1, %rdi
	callq	_malloc
	movq	%rax, -16(%rbp)
	cmpq	$0, -16(%rbp)
	je	LBB0_3
## %bb.2:
	movq	-16(%rbp), %rdi
	movq	-8(%rbp), %rsi
	movq	$-1, %rdx
	callq	___strcpy_chk
LBB0_3:
	jmp	LBB0_4
LBB0_4:
	movq	-16(%rbp), %rax
	addq	$32, %rsp
	popq	%rbp
	retq
	.cfi_endproc
                        ## -- End function
	.globl	_main                   ## -- Begin function main
	.p2align	4, 0x90
_main:                                  ## @main
	.cfi_startproc
## %bb.0:
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	subq	$16, %rsp
	movl	$0, -4(%rbp)
	leaq	L_.str(%rip), %rdi
	leaq	_main.m(%rip), %rsi
	movb	$0, %al
	callq	_printf
	leaq	_main.m(%rip), %rdi
	callq	_strdup
	movq	%rax, -16(%rbp)
	cmpq	$0, -16(%rbp)
	jne	LBB1_2
```

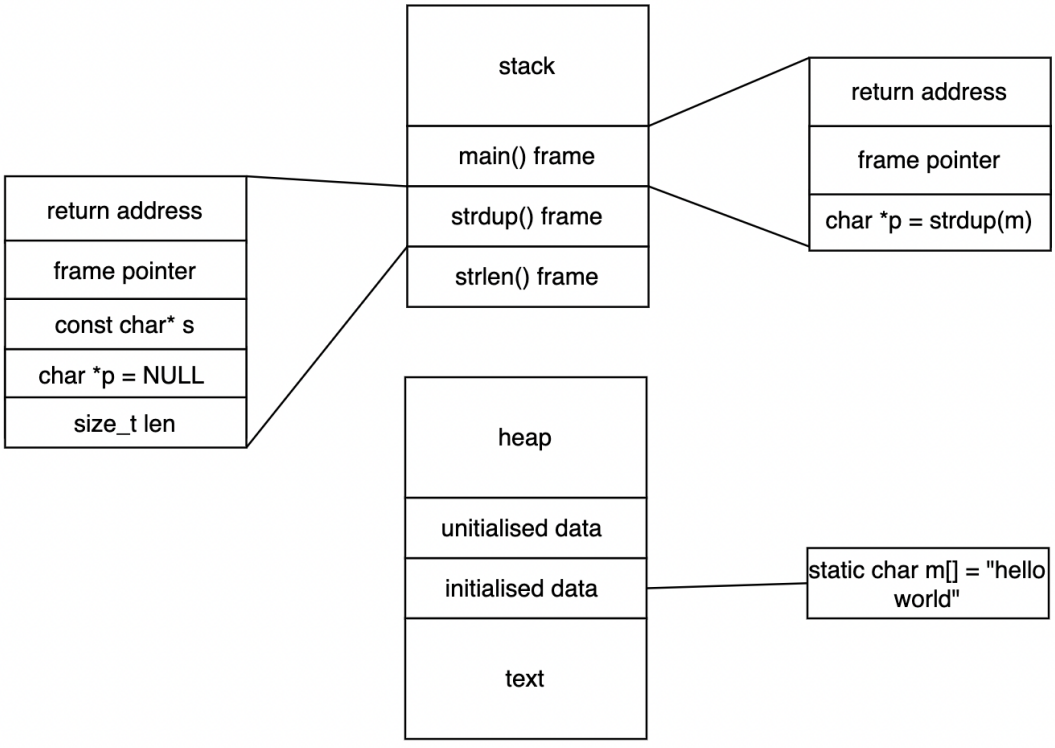
```
## %bb.1:
	leaq	L_.str.1(%rip), %rdi
	callq	_perror
	movl	$1, -4(%rbp)
	jmp	LBB1_7
LBB1_2:
	movq	-16(%rbp), %rdi
	callq	_puts
	cmpl	$-1, %eax
	jne	LBB1_4
## %bb.3:
	leaq	L_.str.2(%rip), %rdi
	callq	_perror
	movl	$1, -4(%rbp)
	jmp	LBB1_7
LBB1_4:
	movq	___stdoutp@GOTPCREL(%rip), %rax
	movq	(%rax), %rdi
	callq	_fflush
	cmpl	$-1, %eax
	jne	LBB1_6
## %bb.5:
	leaq	L_.str.3(%rip), %rdi
	callq	_perror
	movl	$1, -4(%rbp)
	jmp	LBB1_7
LBB1_6:
	movl	$0, -4(%rbp)
LBB1_7:
	movl	-4(%rbp), %eax
	addq	$16, %rsp
	popq	%rbp
	retq
	.cfi_endproc
                                ## -- End function
	.section	__DATA,__data
_main.m:                                ## @main.m
	.asciz	"Hello World!"

	.section	__TEXT,__cstring,cstring_literals
L_.str:                                 ## @.str
	.asciz	"%s"
```

```
L_.str.1:                       ## @.str.1
        .asciz    "strdup"

L_.str.2:                       ## @.str.2
        .asciz    "puts"

L_.str.3:                       ## @.str.3
        .asciz    "fflush"

.subsections_via_symbols

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

char *strdup(const char *s){
        char *p = NULL;

}
Int main(){
        static char m[] = "Hello World!";
        char *p = strdup(m);
        if (!p) {
                perror("strdup");
                return EXIT_FAILURE;
         }
        if (puts(p) == EOF) {
                perror("puts");
                return EXIT_FAILURE;
         }
        if (fflush(stdout) == EOF) {
                perror("fflush");
                return EXIT_FAILURE;
         }
        return EXIT_SUCCESS;
}
```

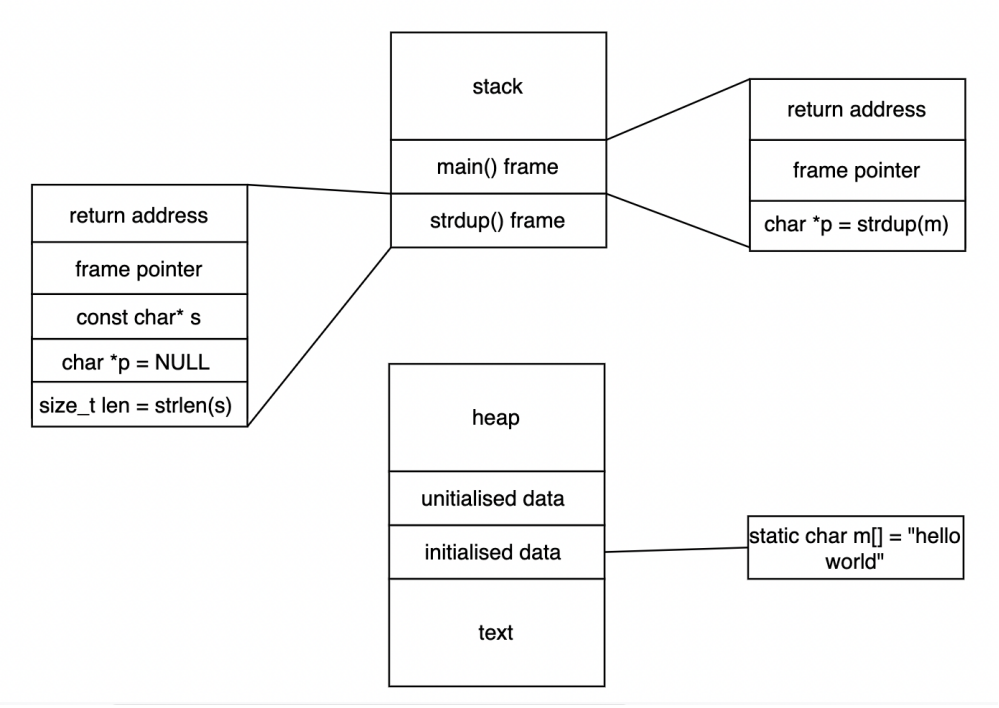
CONTINUES ON NEXT PAGE



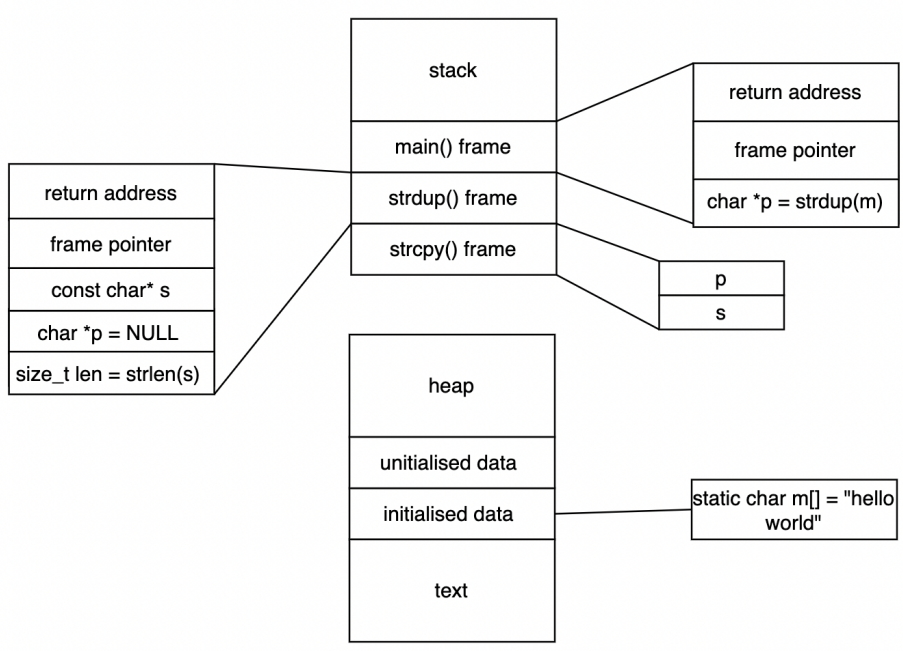
As we enter the main() function a new frame is added to the stack along with a frame pointer pointing to the stack and return address. Since static char m[] is a static variable it will be in the initialized data segment. char *p is a local automatic variable within the main function and will be allocated inside the stack frame. Since p points to strdup(m), the strdup() function call adds a new strdup frame to stack along with const char *s and frame pointer. In the strdup() frame, char* p is a local variable initialized to NULL and is stored in the frame. Size_t len is also a local variable stored in the frame.



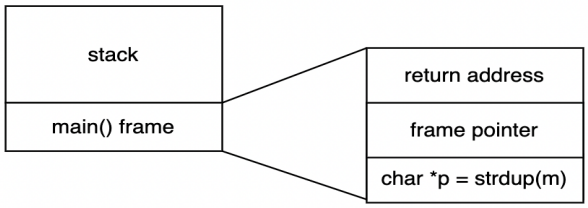
We enter the if condition. If s exists, len = strlen(s). This strlen() function call adds a strlen() frame to stack. When it returns an integer, this frame is popped from stack and this integer is stored in len. p is dynamically allocated to memory in the heap of size len+1.

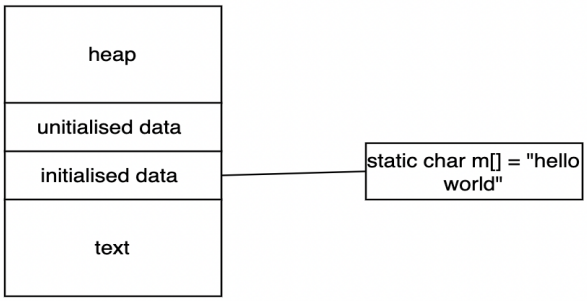


We enter new if condition. If p exists strcpy() frame is pushed onto stack along with variable p and s.



When it returns a value of string p, the strcpy() frame is popped from stack. The value of p is returned and the strdup() frame is popped from stack leaving only main() frame in stack.





We continue in the stack from line 24, checking for the value of p. When main returns a value its frame is popped from the stack.