

CMSC216: Practice Exam 1A SOLUTION

Spring 2024

University of Maryland

Exam period: 20 minutes Points available: 40 Weight: 0% of final grade

Background: Noel Pwanter is experimenting with a linked list application using code shown nearby. The function `list_init()` takes a pointer to a `list_t` struct as its argument. Noel thinks the OLD VERSION for this (show commented) could be improved by directly declaring a pointer as shown in the NEW VERSION. Noel is surprised when Valgrind identifies problems and the program crashes.

<pre> 1 #include "list.h" 2 3 int main(int argc, char *argv[]){ 4 // ... 5 // list_t list; // OLD VERSION 6 // list_init(&list); 7 list_t *listptr; // NEW VERSION 8 list_init(listptr); 9 // ... 10 return 0; 11 } 12 13 // initialize list 14 void list_init(list_t *list){ 15 list->head = NULL; 16 list->size = 0; 17 return; 18 }</pre>	<pre> 1 >> gcc -g list_main.c 2 >> valgrind ./a.out 3 ==4529== Use of uninitialised value of size 8 4 ==4529== at 0x109147: list_init (list_main.buggy.c:15) 5 ==4529== by 0x109133: main (list_main.buggy.c:8) 6 ==4529== 7 ==4529== Invalid write of size 8 8 ==4529== at 0x10914F: list_init (list_main.buggy.c:15) 9 ==4529== by 0x10913B: main (list_main.buggy.c:8) 10 ==4529== Address 0x0 is not stack'd, malloc'd or free'd 11 ==4529== 12 ==4529== Process terminating with default action of 13 ==4529== signal 11 (SIGSEGV): dumping core 14 ==4529== Access not within mapped region at address 0x0 15 ==4529== 16 ==4529== HEAP SUMMARY: 17 ==4529== in use at exit: 0 bytes in 0 blocks 18 ==4529== total heap usage: 0 allocs, 0 frees</pre>
---	---

Problem 1 (15 pts): Answer the following questions about Noel's code.

(A) Describe the problems that Valgrind identifies and what they mean about Noel's code.

SOLUTION: The pointer `listptr` has not been initialized and based on the Valgrind output is defaulting to address 0x0 (NULL). This triggers an out of bounds access and a segmentation fault.

(B) Reverting the code to the OLD VERSION will fix this problem. Describe in which Logical Region of memory the `list` is allocated in the OLD VERSION and WHEN the memory associated with `list` will be de-allocated.

SOLUTION: The old version allocates `list` in the stack where it has enough space for the struct itself. This memory will be de-allocated when `main()` returns like all the other local variables in `main()`'s stack frame.

(C) Noel wants her `listptr` as a pointer to the `list` struct to avoid needing to use the `&list` syntax at later points in her code. What code should she write to achieve this? Indicate if your answer would keep the memory for the `list_t` struct in the same place as the OLD VERSION or move it to a different logical region of memory.

*SOLUTION: Noel could uncomment the OLD VERSION and connect her pointer to the existing list via `list_t *listptr = &list;` This would keep the struct in the stack but provide a pointer to it. The memory would still be de-allocated when `main()` returns.*

*ALTERNATIVELY Noel could heap-allocate the list: keep the OLD VERSION commented and use `list_t *listptr = malloc(sizeof(list_t));`*

to get space in the heap. She would also then need to add `free(listptr)` at the end of her `main()` function to avoid a memory leak.

Problem 2 (15 pts): Nearby is a description of the function `equiv_exchange()` along with a `main()` function demonstrating with example calls. Write this function to meet the specification given.

```

1 #include "equiv_exch.h"
2 typedef struct {
3     char x[128];
4     char y[128];
5 } strpair_t;
6
7 int equiv_exchange(strpair_t *strpair);
8 // If the x/y fields are strings of
9 // equal length, swap them and return 1.
10 // Otherwise do nothing and return 0.
11 // CONSTRAINT: does NOT use strcpy() or
12 // memcpy() functions.
13
14 int main(){
15     int ret;
16     strpair_t elrics = {
17         .x="Ed", .y="Al"
18     };
19     ret = equiv_exchange(&elrics);
20     printf("ret:%d x/y: %s %s\n",
21         ret, elrics.x, elrics.y);
22     // ret:1 x/y: Al Ed
23
24     strpair_t side = {
25         .x="Winry", .y="Mustang"
26     };
27     ret = equiv_exchange(&side);
28     printf("ret:%d x/y: %s %s\n",
29         ret, side.x, side.y);
30     // ret:0 x/y: Winry Mustang
31
32     strpair_t homonc = {
33         .x="Lust", .y="Envy"
34     };
35     ret = equiv_exchange(&homonc);
36     printf("ret:%d x/y: %s %s\n",
37         ret, homonc.x, homonc.y);
38     // ret:1 x/y: Envy Lust
39     return 0;
40 }

```

Note CONSTRAINTs: does not use `strcpy()` / `memcpy()`

YOUR CODE HERE

```

1 // Solution
2 #include "equiv_exch.h"
3 typedef struct {
4     char xstr[128];
5     char ystr[128];
6 } strpair_t;
7
8 int equiv_exchange(strpair_t *strpair){
9     int lenx = strlen(strpair->xstr);
10    int leny = strlen(strpair->ystr);
11    if(lenx != leny){
12        return 0;
13    }
14    for(int i=0; i<lenx; i++){
15        char c = strpair->xstr[i];
16        strpair->xstr[i] = strpair->ystr[i];
17        strpair->ystr[i] = c;
18    }
19    return 1;
20 }

```

Problem 3 (10 pts):

Fill in the following table of equivalent ways to write these 8 bit quantities. There are a total of 9 blanks to fill in and the first column indicates which blanks occur in which lines. Assume two's complement encoding for the signed decimal column.

SOLUTION					Unsigned	Signed
Blank #s	Binary	Hex	Octal	Decimal	Decimal	
#1 #2 #3	0001 1011	0x1B	0033	27	27	
#4 #5 #6	1010 0101	0xA5	0245	165	-91	
~x + 1	0101 1011					
#7 #8 #9	1100 0111	0xC7	0307	199	-57	
~x + 1	0011 1001					

NOTES

- Octal shows leading 0 which is not strictly necessary
- Typical twos' complement conversion technique show below
binary representation: invert bits and add 1