CS-2110 A/B/C Quiz 5 (C)

Aaryan Vinay Potdar

TOTAL POINTS

72 / 100

QUESTION 1

1 1a **2 / 2**

✓ **+ 2 pts** *Correct (14)*

+ 1 pts Treated char* as just char (11)

+ 0 pts incorrect

QUESTION 2

2 1b **2 / 4**

+ 4 pts Both Correct

y1 = 10

y2 = 19

+ 2 pts y1 correct (10)

✓ **+ 2 pts** *y2 correct (19)*

+ 0 pts incorrect

QUESTION 3

3 1c **14 / 14**

✓ **+ 14 pts** *Fully Correct*

{2, 2, 5, 2, -4, 0, 1}

+ 2 pts a correct (2)

+ 2 pts b correct (2)

+ 2 pts c correct (5)

+ 2 pts d correct (2)

+ 2 pts e correct (-4)

+ 2 pts f correct (0)

+ 2 pts g correct (1)

+ 0 pts incorrect

QUESTION 4

4 1d **1 / 10**

+ 10 pts Fully Correct

j = 9

✓ **+ 1 pts** *sets a register value to 9*

+ 2 pts stores value at correct location (R5 - 1)

k = *buf

+ 1 pts loads value of buf into register (R5 + 0)

+ 1 pts loads value at address buf correctly

+ 1 pts Stores value at k correctly (R4 + 0)

i = j - i

+ 1 pts loads i correctly (R4 + 1)

+ 1 pts loads j correctly (R5 - 1)

+ 1 pts Negates i and adds to j (propagate error)

+ 1 pts correctly stores value at i (R4 + 1)

+ 0 pts incorrect

QUESTION 5

5 1ei **2 / 2**

✓ **+ 1 pts** *Correct (False)*

✓ **+ 1 pts** *Valid Explanation:*

- Swap only modifies the copies of x and y on the stack that were passed into the subroutine.

- Need to pass in pointers to correctly modified x and y

+ 0 pts incorrect

QUESTION 6

6 1eii **2 / 2**

✓ **+ 1 pts** *Correct (False)*

✓ **+ 1 pts** *Valid Explanation:*

- `p-- = p - 1* sizeof (*p)`

+ 0 pts incorrect

QUESTION 7

7 1eiii **2 / 2**

✓ **+ 1 pts** *Correct (False)*

✓ **+ 1 pts** *Valid Explanation:*

- C compiler does not do bounds checks

+ 0 pts incorrect

QUESTION 8

8 1eiv **0 / 2**

+ 1 pts Correct (False)

+ 1 pts Valid Explanation:

- You cannot free pointer that has already been freed

- Freeing a pointer twice leads to undefined behavior

✓ **+ 0 pts** *incorrect*

QUESTION 9

9 1ev **0 / 2**

+ 1 pts Correct (False)

+ 1 pts Valid Explanation:

- C code is compiled into assembly which is then assembled into object files

✓ **+ 0 pts** *incorrect*

QUESTION 10

10 1evi **0 / 2**

+ 1 pts Correct (False)

+ 1 pts Valid Explanation:

- A variable declaration makes space for the variable in memory but the value at that memory location is uninitialized

✓ **+ 0 pts** *incorrect*

QUESTION 11

11 1evii **0 / 2**

+ 1 pts Correct (False)

+ 1 pts Valid Explanation:

- You cannot assume the size of a double will be one byte

- Datatypes have different sizes on different OS

✓ **+ 0 pts** *incorrect*

QUESTION 12

12 1eviii **0 / 2**

+ 1 pts Correct (False)

+ 1 pts Valid Explanation:

- You cannot free memory on the stack

✓ **+ 0 pts** *incorrect*

QUESTION 13

13 1eix **1 / 2**

✓ **+ 1 pts** *Correct (False)*

+ 1 pts Valid Explanation:

- String literals are fixed and cannot be modified.

✓ **+ 0 pts** *incorrect*

QUESTION 14

14 1ex **1 / 2**

✓ **+ 1 pts** *Correct (False)*

+ 1 pts Valid Explanation:

- Only compares the addresses of the strings rather than determining if the characters within

those arrays are equivalent

- need to use strcmp

+ 0 pts incorrect

strcmp takes in pointers s1,s2 not the dereferenced values

QUESTION 15

15 2a **0 / 5**

+ 5 pts Correct

- `dealership>head = NULL;`

OR

- `(*dealership).head = NULL;`

OR

- initialized dummy node as head

`Car sentinel = malloc(sizeof(Pet));

sentinel->model = NULL;

sentinal->year= NULL;

sentinal->next = NULL;

dealership->head = sentinel`

- 2 pts minor error

✓ **+ 0 pts** *incorrect*

QUESTION 16

16 2b **15 / 15**

✓ **+ 15 pts** *Correct*

ex))

` Car newCar = malloc(sizeof(Car));*

strcpy(newCar>model, model);

newCar->year = year;

newCar->next = dealer->head;

dealer->head = newCar;

`

OR (w/ sentinel)

` Car newCar = malloc(sizeof(Car));*

strcpy(newCar>model, model);

newCar->year = year;

newCar->next = dealer->head->next;

dealer->head->next = newCar;

`

Create new car

+ 5 pts properly malloc's space for new car

+ 3 pts minor mistake when mallocing space for new car

Assigning model

+ 4 pts uses strcpy to assign model

+ 2 pts minor mistake using strcpy

+ 1 pts assigns model without deep copying

+ 2 pts correctly assigns year

+ 2 pts sets next to first Car in the list correctly

+ 2 pts sets dealership head or sentinel node.next to new car

- 1 pts minor syntax error

+ 0 pts incorrect

- 2 pts Misc. error / incorrect memory management, e.g.

- invalid free
- frees when not necessary
- major syntax error
- undefined behavior

QUESTION 17

17 2c **15 / 15**

✓ **+ 15 pts** *Fully Correct*

Ex))

*`Car *curr = dealership->head;*

while (curr != NULL) {

printf("%s (year %d)/n", curr->model, curr->year);

curr = curr->next;

}

`

OR (w/sentinel)

*`Car *curr = dealership->head;*

while (curr->next != NULL) {

printf("%s (year %d)/n", curr->next->model, curr->next->year);

curr = curr->next;

}

`

+ 2 pts creates a variable to loop through list (ex. curr) and initializes to first node

+ 2 pts loop structure

+ 2 pts correct end condition (curr == NULL)

printf statement

+ 6 pts fully correct

+ 4 pts minor syntax or formatting error

ex))

- forget /n

- switches model and year

- uses dot op instead of arrow

+ 2 pts major error but attempts a printf statement with car's model and year passed into the function

+ 3 pts incrementing condition to move through the list

- 1 pts minor syntax error not accounted for in rubric

+ 0 pts incorrect

QUESTION 18

18 2d **15 / 15**

✓ **+ 15 pts** *Fully Correct*

ex))

`

Car curr = dealership->head;*

while (curr) {

*Car *next = curr->next;*

free(curr);

curr = next;

}

free(dealership)

`

+ 2 pts correctly creates a variable to loop through list (ex. curr) and initializes to first node

+ 2 pts structure to loop through cars

+ 2 pts correct terminating condition when end of list is reached

+ 3 pts temp variable for storing curr->next BEFORE freeing curr

+ 3 pts frees the current car within loop

+ 3 pts frees the dealership AND does not try to access it after freeing

- 2 pts frees care model AND/OR year

- 2 pts using * incorrectly

- 2 pts uses var after freeing

- 2 pts Used `malloc` on `curr` and/or `temp`

- 1 pts minor syntax error

+ 0 pts incorrect

good

Your Initials: AP

Name [PRINT CLEARLY]: AARYAN POTDAR

GT username (e.g. gburdell3): apotdar31

CS 2110: Computer Organization and Programming
Conte/Gupta/Adams Fall 2023

QUIZ 5
VERSION C

This exam is given under the Georgia Tech Honor Code System. Anyone found to have submitted copied work instead of original work will be dealt with in full accordance with Institute policies.

Georgia Tech Honor Pledge: *"I commit to uphold the ideals of honor and integrity by refusing to betray the trust bestowed upon me as a member of the Georgia Tech community."*

[MUST sign:] AP

- THIS IS A CLOSED BOOK, CLOSED NOTES EXAM
- NO CALCULATORS
- This examination handout has **8** pages.
- Do all your work in this examination handout.
- Only the front of exams sheets will be scanned. Do **not** write your answer on the back of the exam sheets.
- Please write your initials at the top of each page
- WHERE NEEDED, SHOW ALL YOUR INTERMEDIATE RESULTS TO RECEIVE FULL CREDIT

In case you forgot, here are some good facts to know:

Hex	Dec
0x1	1
0x2	2
0x3	3
0x4	4
0x5	5
0x6	6
0x7	7
0x8	8
0x9	9
0xA	10
0xB	11
0xC	12
0xD	13
0xE	14
0xF	15

x	2^x
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16,384
15	32,768
16	65,536

Problem	Points	Score
1	50	
2	50	
TOTAL	100	

GOOD LUCK!

More good facts to know:

$1K = 2^{10}$
$1M = 2^{20}$
$1G = 2^{30}$
$1T = 2^{40}$
$1P = 2^{50}$
$1E = 2^{60}$

Your Initials: AP

1. [50 pts] Answer the following short questions. Show your work (where needed) to receive full c

(a) Fill in the result when the following C code is executed. Assume an address is 32 bits and a char is one byte. (*If needed, you may assume the compiler does not add any padding to the struct.*)

```
struct our_st {
    char *p;
    char x[10];
} b;

printf("Size is %d\n", sizeof(b));
```

Prints: Size is 14

(b) Fill in the result when the following C code is executed.

```
#define Y(X) m*X+b
int main(void) {
    int m = 4;
    int b = 3;
    int y1 = Y(1+3);
    int y2 = Y((1+3));
    printf("y1 is %d\n", y1);
    printf("y2 is %d\n", y2);
}
```

Prints: y1 is 16

Prints: y2 is 19

(c) Compute the results of the following C expressions. Write the computed value as a C integer constant (e.g., -2, 0x35, etc.). You may assume C integers use two's complement.
Note: x and y are ints, and the code x = 2 and y = -2 runs before each expression (*i.e., the expressions below are independent from one another*)

C Expression	Result
`a = ((x > y)? 4: 32)+y`	a is 2
`b = y = x`	b is 2
`c = (x++) - (--y)`	c is 5
`d = x & y`	d is 2
`e = y << (--x)`	e is -4
`f = y == x`	f is 0
`g = x && y`	g is 1

Your Initials: AP

(d) Compile the C code below between ***BEGIN*** and ***END*** into LC-3 assembly assuming the LC-3 memory layout is used (*i.e., R4 is the global pointer and R5 is the local pointer*). Also assume all C ints and addresses are ~~16 bits~~ when in LC-3:

```
int k;
int i;
int main(void) {
    int *buf;
    int j;
    buf = malloc(128 * sizeof(int));
    // ...
    // *BEGIN*
    j = 9;
    k = *buf;
    i = j - i;
    // *END*
    // ...
}
```

```
AND R0, R0, 0
LDR R0, R5, #-1 ;; load j local variable.
ADD R0, R0, 9   ;; set j = 9.
LDR R1, R4, #0  ;; load k global var
LDI R1, R2, #0  ;; k = *buf;
LDR R3, R4, #1  ;; load global variable i
NOT R3, R3, 0   ;; i = -i
ADD R3, R3, 1   ;; Negate i
ADD R3, R0, R3  ;; i = j - i
STR R0, R5, #-1
STR R3, R4, #1
```

Ans:

```
LDR R0, R5, #-1
AND R0, R0, 0
ADD R0, R0, 9
LDR R1, R4, #0
LDR R2, R5, #0
LDI R1, R2, #0
NOT R3, R3, 0
ADD R3, R3, 1
ADD R3, R0, R3
STR R0, R5, #-1
STR R3, R4, #1
STI R2, R5, #0
```

= Answer written in box. (please ignore the pencil).

Your Initials: AP

(e) Answer the following true/false questions by circling "true" or "false," and then give a reason for each answer:

TRUE or ~~FALSE~~	Assume the following definition of swap(): `void swap(int x, int y) { int tmp = x; x = y; y = tmp; }` Then I can write **`swap(a, b)`** to swap the values of two **`int`** variables **`a`** and **`b`**. Why or why not? False. The values of the integers get swapped locally inside the function. If x & y are global variables, they retain their original values.
TRUE or ~~FALSE~~	If **`p`** is a pointer, then **`p--`** will always decrement the address held in **`p`** by 1 memory location. For example, if **`p`** starts as **0x500F**, **`p--`** will always change **`p`** to **0x500E**. Why or why not? Pointer arithmatic will implicitly account for data type ∴ (-1 * sizeof(datatype). Also if p is a constant pointer, p-- re-assignment is not allowed in C.
TRUE or ~~FALSE~~	Accidentally writing past the end of an array (a "buffer overflow") is uncommon in C because the compiler inserts bounds checks for every array access. Why or why not? Will result in segmentation fault, causing runtime - errors.

~~TRUE~~ or FALSE	In C, it is *good* practice to **`free()`** a pointer multiple times to avoid memory leaks, as shown below:
	`int *ptr = malloc(/* ... */);`
	`free(ptr);`
	`free(ptr); // just in case`
	Why or why not?
	Free-ing a null pointer can avoid memory leaks in case of malloc failure. We can free the pointer twice. Free a null pointer has no effect but doesn't throw error.

TRUE TRUE or ~~FALSE~~	There is no assembler involved in the process of compiling a program consisting of many .c files to an executable. Why or why not? A linker program is used to compile multiple object files into executable. Two-pass assemble is used in preparing the object file.

Your Initials: AP

TRUE or FALSE	A variable declaration in a function initializes memory to zero by default. Why or why not? ~~Setting the value to zero~~ for local variable defined inside function, default value is initialized to 0!
TRUE or FALSE	The following C code will allocate enough space for an array of 18 doubles regardless of the compiler or target architecture: `double *array = malloc(18);` Why or why not? the malloc will return a (void *) pointer which will be sufficiently large. the compiler will automatically cast to (double*) after assignment. (unexpected casts are bad practises in theory). malloc will return (18*8) → same for double.

TRUE or FALSE	In C, you should `free()` *any* memory you are no longer using, including stack variables. That is, this code shows good programming practice:
	`{`
	`  int j = 24;`
	`  // ...`
	`  free(&j); // j falls out of scope, free it`
	`}`
	Why or why not?
	It's a good practise to avoid memory leaks in C. local variables are stored on stack. although dynamic, it has a physical limit.

TRUE or FALSE	Like Java or Python, C string literals can be concatenated with +, like `"hi, " + "dennis!"`. Why or why not? A terminator character is automatically added after the string within " ", so cannot concatenate like that. must use a string function.
TRUE or FALSE	Suppose you have two C strings `s1` and `s2`; (specifically, you wrote `char *s1, *s2;` already and initialized both variables). Then you can determine if `s1` and `s2` point to character arrays that contain the same characters with the expression `s1 == s2`. Why or why not? We must use strcmp(*s1, *s2) to compare the two strings. it will return 0 if the two strings are equal based on ASCII value comparison.

Your Initials: AP

2. [50 pts] The following car dealership inventory code makes use of a linked list of cars, which you will partially implement per the comments below.
You may assume that malloc never returns NULL, and no functions are ever passed a NULL "`dealer`" or "`model`".

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

typedef struct car {
  char model[32];
  int year;
  struct car *next;
} Car;

struct dealership {
  Car *head;
};

struct dealership *dealership_new(void);
void dealership_push_front(struct dealership *, char *, int);
void dealership_print(struct dealership *);
void dealership_destroy(struct dealership *);

int main(void) {
  struct dealership *dealer = dealership_new();
  dealership_push_front(dealer, "Canyon", 2015);
  dealership_push_front(dealer, "Silverado", 2006);
  dealership_push_front(dealer, "El Camino", 1986);

  dealership_print(dealer);
  dealership_destroy(dealer);
}


struct dealership *dealership_new(void) {
  struct dealership *dealer = malloc(sizeof (struct dealership));

  // Initialize the newly allocated `dealer' to be empty
```

dealer -> Car -> head = NULL; // empty linked list

```
  return dealer;
}
// Continues on next page
```

```
// Continued from last page

void dealership_push_front(struct dealership *dealer, char *model, int year) {
  // Add a new node (car) to the beginning of the linked list (dealer).
  // The new node's model and year should be the arguments `model' and `year',
  // respectively
  // You are allowed to use functions declared in string.h, like strlen(),
  // strcpy(), etc. Assume model != NULL and is less than 32 characters
  // (including the null terminator)
```

```
Car *newCar = (*Car) malloc (sizeof (Car));
newCar -> year = year;
strcpy (newCar -> model, model);
if ( dealer -> head == null) {   // empty list.
 dealer -> head = newCar;
  newCar -> next = NULL;
} else {
    newCar -> next = dealer -> head;
    dealer -> head = newCar;
}
```

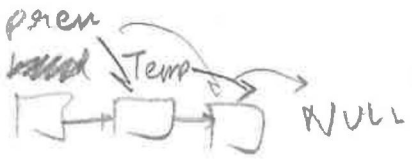
```
}

void dealership_print(struct dealership *dealer) {
  // Print all cars at the dealership, one per line.
  // Each car should be printed in the format
  //     Canyon (year 2015)
  //     Silverado (year 2006)
  //     El Camino (year 1986)
  // etc. Don't forget the newline!
```

```
Car * curr;
curr = dealer -> head;
if(dealer == NULL){
  return
}
while (curr != NULL) {
    char *name;
    strcpy (name, curr -> model)
    int year = curr -> year
    printf ("%.S %d)\n", name,
                          year);
    curr = curr -> next;
}
```

```
}
// Continues on next page
```

Your Initials: AP




```
// Continued from last page

void dealership_destroy(struct dealership *dealer) {
  // To avoid leaking memory, destroy the linked list `dealer'. This means to
  // free() all nodes plus free()ing the list itself.
  // Watch out not to use data after you free() it!
```

~~Car * curr;~~
~~Car * next;~~
~~curr = dealer -> head;~~
~~while (curr != NULL);~~

~~while~~

```
Car * prev = dealer -> head;
Car * temp = NULL;
temp = prev -> next;
while (temp != NULL) {
    free (prev);
    prev = temp;
    temp = temp -> next;
}
free(prev); // if list only had one Node.
free (dealer); // free list itself.
}
```