Quiz 5

● Graded

Student

[redacted]

Total Points

77 / 100 pts

Question 1

1a **1** / 2 pts

+ 2 pts Correct (18)

✔ **+ 1 pt** Treated char* as a char (11)

+ 0 pts incorrect

Question 2

1b **2** / 4 pts

+ 4 pts Both Correct
y1 = 9
y2 = 17

+ 2 pts y1 correct (9)

✔ **+ 2 pts** y2 correct (17)

+ 0 pts incorrect

Question 3

1c **14** / 14 pts

✔ **+ 14 pts** Fully Correct

✔ **+ 2 pts** a correct (0)

✔ **+ 2 pts** b correct (2)

✔ **+ 2 pts** c correct (-1)

✔ **+ 2 pts** d correct (1)

✔ **+ 2 pts** e correct (-1)

✔ **+ 2 pts** f correct (-2)

✔ **+ 2 pts** g correct (1)

+ 0 pts incorrect

Question 4

1d **2** / 10 pts

+ 10 pts Fully Correct

---

y = -1

✔ **+ 1 pt** sets a register value to -1

+ 2 pts stores -1 at correct location (R5 + 0)

---

z = *p

+ 1 pt loads value of p into register correct (R5-1)

+ 1 pt loads value at address p correctly

+ 1 pt Stores *p at z correctly (R4 + 1)

---

x = x&y

+ 1 pt loads x correctly (R4 + 0)

+ 1 pt loads y correctly (R5 + 0)

✔ **+ 1 pt** ANDs x & y (propagate error)

+ 1 pt correctly stores ANDed value at x (R4 + 0)

+ 0 pts incorrect

Question 5

1ei **2** / 2 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 1 pt** Valid Explanation:

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

+ 0 pts incorrect

Question 6

1eii **2** / 2 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 1 pt** 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 7

1eiii **1** / 2 pts

✔ **+ 1 pt** Correct (False)

+ 1 pt Valid Explanation:

- String literals are fixed and cannot be modified.

+ 0 pts incorrect

Question 8

1eiv **1** / 2 pts

✔ **+ 1 pt** Correct (False)

+ 1 pt Valid Explanation:

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

+ 0 pts incorrect

Question 9

1ev **2** / 2 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 1 pt** 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 10

1evi **2** / 2 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 1 pt** Valid Explanation:

- C compiler does not do bounds checks

+ 0 pts incorrect

Question 11

1evii **0** / 2 pts

+ 1 pt Correct (False)

+ 1 pt Valid Explanation:

- Only compares the addresses of the strings rather than determining if the characters within those arrays are equivalent

✔ **+ 0 pts** incorrect

Question 12

1eviii **2** / 2 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 1 pt** Valid Explanation:

- You cannot free memory on the stack

+ 0 pts incorrect

Question 13

1eix **2** / 2 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 1 pt** Valid Explanation:

- You cannot free pointer that has already been freed
- Freeing a pointer twice leads to undefined behavior

+ 0 pts incorrect

Question 14

1ex **2** / 2 pts

✔ **+ 1 pt** Correct (False)

✔ **+ 1 pt** 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 15

2a **0** / 5 pts

+ 5 pts Correct

- `store->head = NULL;`
  OR
- initialized dummy node as head
  `Pet *sentinel = malloc(sizeof(Pet));`
  `sentinel->name = NULL;`
  `sentinel->age = NULL;`
  `sentinel->next = NULL;`
  `store->head = sentinel`

– 2 pts minor error

✔ **+ 0 pts** incorrect

Question 16

2b **15** / 15 pts

✔ **+ 15 pts** Correct

ex))

```
Pet* newPet = malloc(sizeof(Pet));
  strcpy(newPet->name, name);
  newPet->age = age;
  newPet->next = store->head;
  store->head = newPet;
```

OR (w/ sentinel)

```
Pet* newPet = malloc(sizeof(Pet));
  strcpy(newPet->name, name);
  newPet->age = age;
  newPet->next = store->head.next;
  store->head.next = newPet;
```

Create new pet

+ 5 pts properly malloc's space for new pet

+ 3 pts minor mistake when mallocing space for new pet

Assigning name

+ 4 pts uses strcpy to assign name

+ 2 pts minor mistake using strcpy

+ 1 pt assigns name without deep copying

+ 2 pts correctly assigns age

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

+ 2 pts sets store head or sentinel node.next to new pet

– 1 pt minor syntax error

+ 0 pts incorrect

Question 17

2c **15** / 15 pts

✔ **+ 15 pts** Fully Correct

Ex))

```
Pet *curr = store->head;
while (curr != NULL) {
  printf("%s (age %d)/n", curr->name, curr->age);
  curr = curr->next;
}
```

OR (w/ sentinel)

```
Pet *curr = store->head;
while (curr->next != NULL) {
  printf("%s (age %d)/n", curr->next->name, curr->next->age);
  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 name and age
- uses dot op instead of arrow

+ 2 pts major error but attempts a printf statement with pet's name and age passed into the function

---

+ 3 pts incrementing condition to move through the list

− 1 pt minor syntax error not accounted for in rubric

+ 0 pts incorrect

Question 18

2d **12** / 15 pts

+ 15 pts Fully Correct

ex))

```
Pet* curr = store->head;
while (curr) {
  Pet *next = curr->next;
  free(curr);
  curr = next;
}
free(store)
```

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

✔ **+ 2 pts** structure to loop through pets

✔ **+ 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 Pet within loop

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

− 2 pts frees pet name AND/OR AGE

− 2 pts using * incorrectly

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

− 2 pts Undefined behavior

− 1 pt minor syntax error

+ 0 pts incorrect

Your Initials: [redacted]

Name [PRINT CLEARLY]: [redacted]

GT username (e.g. gburdell3): [redacted]

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

QUIZ 5
VERSION A

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:] [redacted]

- 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: _____

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

(a) Fill in the result when the following C code is executed. Assume an address is 64 bits and a char is one byte. (*If needed, you may assume the compiler does not add any padding to the struct.*)

```
struct my_st {
    char *z;
    char a[10];
} i;

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

Prints: Size is 11

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

```
#define Y(X) m*X+b
int main(void) {
    int m = 5;
    int b = 2;
    int y1 = Y(1+2);
    int y2 = Y((1+2));
    printf("y1 is %d\n", y1);
    printf("y2 is %d\n", y2);
}
```

Prints: y1 is 7

Prints: y2 is 17

(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 = 0` and `y = -1` runs before each expression (*i.e., the expressions below are independent from one another*)

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

Your Initials: _____

(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 x; R4 = 0
int z; R4 = 1
int main(void) {
    int y; R5 = 0
    int *p; R5 = -1
    p = malloc(3 * sizeof(int));
    // ...
    // *BEGIN*
    y = -1;
    z = *p;
    x = x & y;
    // *END*
    // ...
}
```

```
AND R0, R0, 0 ; // y
AND R1, R1, 0 ; // *p
AND R2, R2, 0 ; // x

ADD R0, R0, -1 ; // y = -1
STR R0, R5, 1 ; // mem[y] = -1

LDR R1, R5, 0 ; // *p
STR R1, R4, 0 ; // z = *p

LDR R2, R4, -1 ; // x
AND R2, R2, R0 ;
STR R2, R4, -1 ;
```

Your Initials: _____

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

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? False because the assembler is needed to properly convert the C code into machine level code.
TRUE or FALSE	A variable declaration in a function initializes memory to zero by default. Why or why not? No because in a function, a variable initializes to undesired behavior or garbage
TRUE or FALSE	Like Java or Python, C string literals can be concatenated with +, like **`"hello, " + "world!"`**. Why or why not? No because
TRUE or FALSE	If **`p`** is a pointer, then **`p++`** will always increment the address held in **`p`** by 1 memory location. For example, if **`p`** starts as **0x4000**, **`p++`** will always change **`p`** to **0x4001**. Why or why not? No because array names when reduced to pointers are nonmodifiable.
TRUE or FALSE	Assume the following definition of swap(): `void swap(int a, int b) { int tmp = a; a = b; b = tmp; }` Then I can write **`swap(x, y)`** to swap the values of two **`int`** variables **`x`** and **`y`**. Why or why not? Since this is a function, the values attained will be cleared from the stack once its call finishes since these are mere local variables.
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? The compiler does not check for bounds, so it is up to the programmer to correctly write to a memory address along an array.

Your Initials: _____

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? Yes because strings can be reduced to arrays, so $s_1 = s_2$ would be evaluated if the memory address of the beginning for each string is the same.

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 x = 3;`
	`  // ...`
	`  free(&x); // x falls out of scope, free it`
	`}`
	Why or why not?
	
	No, the only memory that needs to be freed are memory that lies on the heap as they are dynamically allocated unlike the stack.

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?
	
	No because you should never manipulate a freed pointer, or else there will be a segment fault.

TRUE or FALSE	The following C code will allocate enough space for an array of 4 doubles regardless of the compiler or target architecture: `double *array = malloc(4);` Why or why not? No because malloc needs the sizeof in order to correctly allocate. So, double * array = malloc(4 * sizeof(double));

Your Initials: _____

2. [50 pts] The following pet store inventory code makes use of a linked list of pets, 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 pet "**store**" or "**name**".

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

typedef struct pet {
  char name[32];
  int age;
  struct pet *next;
} Pet;

struct pet_store {
  Pet *head;
};

struct pet_store *pet_store_new(void);
void pet_store_push_front(struct pet_store *, char *, int);
void pet_store_print(struct pet_store *);
void pet_store_destroy(struct pet_store *);

int main(void) {
  struct pet_store *store = pet_store_new();
  pet_store_push_front(store, "Pinky", 1);
  pet_store_push_front(store, "Fido", 16);
  pet_store_push_front(store, "Pugsley", 12);

  pet_store_print(store);
  pet_store_destroy(store);
}


struct pet_store *pet_store_new(void) {
  struct pet_store *store = malloc(sizeof (struct pet_store));

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

store = (struct pet_store *) store;

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

Your Initials: _____

// Continued from last page

```
void pet_store_push_front(struct pet_store *store, char *name, int age) {
  // Add a new node (pet) to the beginning of the linked list (store).
  // The new node's name and age should be the arguments `name' and `age',
  // respectively
  // You are allowed to use functions declared in string.h, like strlen(),
  // strcpy(), etc. Assume name != NULL and is less than 32 characters
  // (including the null terminator)
    struct Pet *newNode = (struct Pet*)malloc(sizeof(struct Pet));
    int i=0;
    while(name[i] != '\0'){
      newNode->name[i] = name[i];
      i++;
    }
    newNode->age=age;
    newNode->next = store->head;
    store->head = newNode;
    return;
}

void pet_store_print(struct pet_store *store) {
  // Print all pets in the store, one per line.
  // Each pet should be printed in the format
  //     Pinky (age 1)
  //     Fido (age 16)
  //     Pugsley (age 12)
  // etc. Don't forget the newline!
    struct Pet *current = store->head;

    while(current != null){
      Printf("%s (age %d)\n", current->name, current->age);
      current = current->next;
    }

    return;
}
```

// Continues on next page

Your Initials: _____

```
// Continued from last page

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

```
Struct Pet *Previous = store->head;
Struct Pet * next = store->head->next;
While(next != null){
    free(previous);
    Previous = next;
    next = next->next;
}
free(Previous);
return;
```

```
}
```