Quiz 1 Review with Sofia F.

ANSWER KEY

Date: Monday June 3, 2024

Time and location: 3:00 PM at CB1 117

Hello everyone! As some of you may now, I am the SARC SI Leader designated to Dr. Ahmed's section of CS1. In preparation for Quiz 1 I am doing a quiz review. In this document you can find different questions for most of the topics presented in the quiz. You can try these problems on your own, and on the day of the review we will go over the solutions for all of them together.

DISCLAIMER: don't rely solely on this quiz review as preparation for the quiz. Read Dr. Ahmed's announcement thoroughly and make sure you are preparing for everything listed in there.

Dynamic Memory Allocation (DMA)

Dynamic memory functions:

1. Write a single line of code where you use *malloc* to allocate for an array of n integers. Make sure to declare the necessary variables to store the address of the allocated memory.
   ```
   int *array = malloc(sizeof(int)*n);
   ```

2. Write a single line of code where you use *calloc* to allocate for n integers.
   ```
   int *array= calloc(n, sizeof(int));
   ```

3. Write a single line of code where you use realloc to increase the previously allocated array. Save the reallocated memory appropriately.
   ```
   int *array= realloc(array, 2n*sizeof(int));
   ```

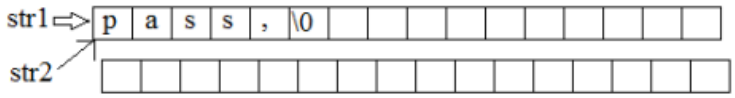
What is the main difference between using malloc and calloc?

Malloc allocates a single block of memory and does not initialize it. Calloc allocates multiple blocks of memory and initializes everything to 0.

How does realloc manage memory?

it first allocates new memory and copies contents, then it frees past memory.

(a) (3 pts) Draw a picture that indicates the relevant state of memory ***after*** line 14 has completed. (Draw a rectangular box to indicate dynamically allocated memory.)



Note: All cells left empty represent uninitialized character variables.

Grading: 1 pt for having two boxes drawn indicating allocated memory. 1 pt for having str1 point to the box that stores "pass," (this must be indicated), 1 pt for having str2 point to this same box.

(b) (1 pt) Explain why line 14 causes a memory leak.

When the pointer str2 moves, nothing is pointing to the memory that it used to be pointing to originally.

(c) (1 pt) Why is it possible for the code to crash on line 21?

str2 is pointing to NULL (nothing), so it's not pointing to dynamically allocated memory. Attempting to free memory that isn't dynamically allocated may crash a program.

Grading parts (b) and (c): Give the point for each if the answer is reasonably close or shows that the student understands the key issue at hand. **No half points! Only award an integer number of points.**

Fall 2016 **Data Structures Exam, Part A**

1) (10 pts) DSN (Dynamic Memory Management in C)

Consider the following struct, which contains a string and its length in one nice, neat package:

```
typedef struct smart_string {
  char *word;
  int length;
} smart_string;
```

Write a function that takes a string as its input, creates a new *smart_string* struct, and stores a **new copy of that string** in the *word* field of the struct and the length of that string in the *length* member of the struct. The function should then return a pointer to that new *smart_string* struct. Use dynamic memory management as necessary. The function signature is:

```
smart_string *create_smart_string(char *str) {

   smart_string *s = malloc(sizeof(smart_string));

   s->length = strlen(str);

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

   strcpy(s->word, str);

   return s;

}
```

Grading (7 pts): 1 pt for smart_string declaration, 1 pt for first malloc, 1 pt for setting length correctly, 1 pt for sizeof(char) in second malloc, 1 pt for (s->length + 1) in second malloc, 1 pt for using strcpy, and 1 pt for correct return statement. They can also use calloc(). Please deduct a point for other large, obvious errors (such as using the . operator instead of the -> operator).

Now write a function that takes a *smart_string* pointer (which might be NULL) as its only argument, frees all dynamically allocated memory associated with that struct, and returns NULL when it's finished.

```
smart_string *erase_smart_string(smart_string *s) {

   if (s != NULL)
   {
      free(s->word); // This is safe, even if word is NULL.
      free(s);
   }
   return NULL;
}
```

Grading (3 pts): 1 pt for checking s != NULL, 1 pt for free(s->word), and 1 pt for free(s).

Linked List

Summer 2018 **Data Structures Exam, Part A**

2) (10 pts) ALG (Linked Lists)

Suppose we have a linked list implemented with the structure below. Write a function that will take in a pointer to the head of list and inserts a node storing -1 after each even value in the list. If the list is empty or there are no even values in the list, no modifications should be made to the list. (For example, if the initial list had 2, 6, 7, 1, 3, and 8, the resulting list would have 2, -1, 6, -1, 7, 1, 8, -1.)

```
typedef struct node {
    int data;
    struct node* next;
} node;

void markEven(node *head) {

    node* tmp = head;

    while (tmp != NULL) {      // 2 pts iter whole list

        while (tmp != NULL && tmp->data%2 != 0) //3pts find next even
            tmp = tmp->next;

        if (tmp != NULL) {                    // 1 pt no null error
            node* newnode = malloc(sizeof(node));
            newnode->data = -1;               // 2 pts make new node
            newnode->next = tmp->next;        // 2 pts patch it into
            tmp->next = newnode;              // list
            tmp = newnode;
        }
    }
}
```

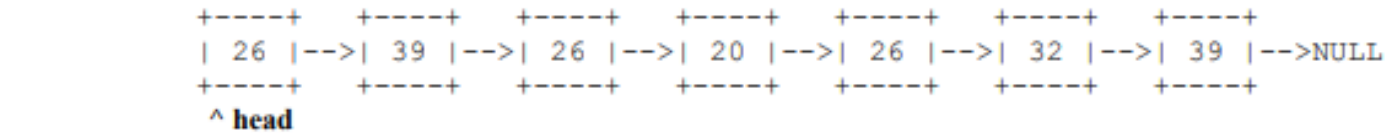
Spring 2017 **Data Structures Exam, Part A**

2) (5 pts) ALG (Linked Lists)

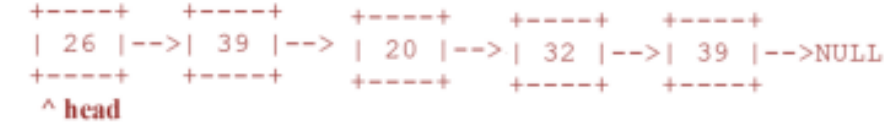
Consider the following function that takes in as a parameter a pointer to the front of a linked list(`list`) and the number of items in the list(`size`). ***node*** is defined as follows:

```
typedef struct node {
    int data;
    struct node* next;
} node;

int mystery(node* list, int size) {
    node* prev = list;
    node* temp = list->next;

    while (temp != NULL) {
        if (list->data == temp->data) {
            prev->next = temp->next;
            free(temp);
            size--;
            temp = prev->next;
        }
        else {
            prev = prev->next;
            temp = temp->next;
        }
    }
    return size;
}
```

If **`mystery(head, 7)`**, is called, where head is shown below, what will the function return and draw a picture of the resulting list, right after the call completes?



Adjusted List



The function returns 5.

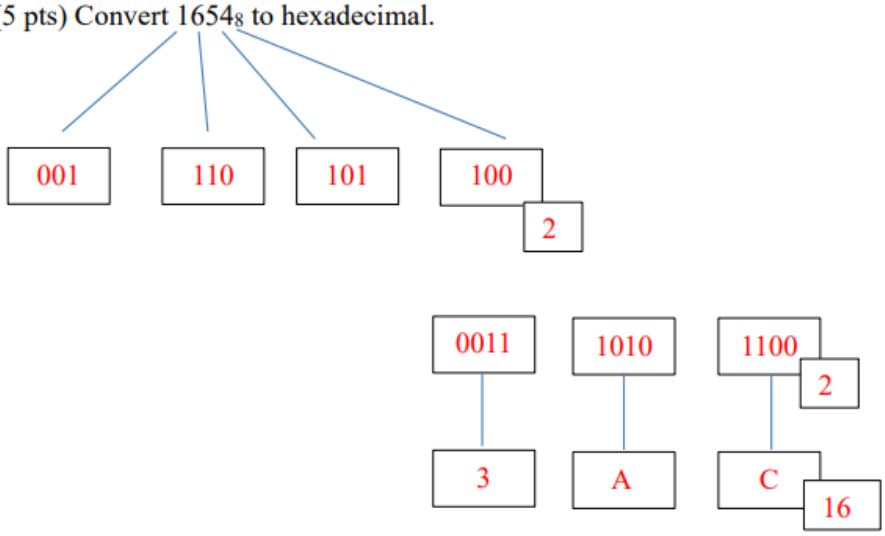
Grading: 2 pts return value (all or nothing), 3 pts list, give partial for list as you see fit.

Base conversion

Summer 2015 **Computer Science Exam, Part A**

5) (10 pts) ALG (Base Conversion)

(a) (5 pts) Convert 1654_8 to hexadecimal.



$3AA_{16}$

Grading: 2 pts for converting from base 8 to base 2, 1 pt for regrouping from sets of three values to sets of four values, 2 pts for converting regrouped base 2 to base 16.

(b) (5 pts) Convert 1925_{10} to octal.

1925/8	=	240	with remainder	5
240/8	=	30	with remainder	0
30/8	=	3	with remainder	6
3/8	=	0	with remainder	3

Answer: $1928_{10} = 3605_8$

Grading: 1 pt for each division by 8 with remainder, 1 pt for final solution in correct order

Grading: 5 pts total - 1 pt for each group of 4 bits. All 4 bits in the group have to be correct to get the point.

b) Frank is the team-lead for the software testing team at his job. He is celebrating his birthday. Some of his co-workers have baked a cake for the celebration and thought that it would be really cool to put candles on his cake to represent his age in binary. An unlit candle represents the 0 bit. From the pic of the cake below, how old is Max?

100111

32 4 2 1

= 32 + 4 + 2 + 1 = 39

Grading: 5 pts total
4 pts : 1 for decimal for each digit
1 pt for final answer

Note: Also give full credit for 32 + 16 + 8 + 1 = 57, though most students will read left to right.

Queues

What is the principle for queues?

What are some situations where queues are used in real life?

Write the enqueue and dequeue functions using the following queue struct declaration:

```
typedef struct Queue
{
  int queue_array[MAX];
  int rear;
  int front;
}Queue;
```

```
void enQueue(Queue *q, int add_item)
{

   if (q->rear <MAX)
   {
       q->queue_array[q->rear++] = add_item;
       printf("\nAdded to queue\n");
   }
   else
   {
       printf("Queue Overflow \n");
   }

}
```

```
int deQueue(Queue *q)
{

   if(q->front==q->rear)
   {
       printf("\n Empty. \n");
       return EMPTY;
   }
   else
       return q->queue_array[q->front++];

}
```