ESC190 Midterm 2

Sample Solution

And Grading Scheme

Question 1a

```
int *appl = &x;
int **bork = &appl;
int ***catt = &bork;
int **doge = *catt;
int ***emuu = &doge;
```

Marking Scheme:
-0.5 per incorrect response

```
int x_copy = ***emuu;
```

Marking Scheme:
-1 if incorrect based on the response to `int ___emuu = &doge;`

Question 1b

```
struct Person{
      int age;
      struct Person *enemy;
};
int main(void){
      struct Person hurry;
      struct Person voldy;
      hurry.age = 16;
      voldy.age = 200;
      hurry.enemy = &voldy;
      voldy.enemy = &hurry;
}
```

Marking Scheme:
-0.5 per incorrect response/bug e.g., forgetting ampersand(s) or semicolon(s), using '->' instead of '.'

Question 2a

Solution 1:

```
#include <stdlib.h>
int main(void){
      int *ptr = malloc(sizeof(int) * 4);
      for(int i = 3; i >= 0; i--){
            *(ptr + i) = i;
            free(ptr + i);
      }
      free(ptr);
}
```

Solution 2:

```
#include <stdlib.h>
int main(void){
      int *ptr = malloc(sizeof(int) * 4);
      for(int i = 3; i >= 0; i--){
            *(ptr + i) = i;
            free(ptr + i);
      }
}
```

Marking Scheme:
+2 for implementing one of the solutions

Question 2b

Need to mention that during compile, no memory access is checked. The variable types and syntax all make sense in the above code. It is only during runtime that the invalid input to free was computed.

Marking Scheme:
+1 for writing the correct explanation (or some valid variant)

Question 3a

Since the question stated that b = n, either option is necessary.

Marking Scheme:
+1 if circled (i)
+1 if circled (ii)
+1 if circled (i) and (ii)
0 if did not circle anything

Question 3b

No matter what load factor you have, you may get lucky and look up from an empty bucket/bucket with one key. You may also get unlucky and look up from a bucket that contains all the keys. Hence, the top row should only have $\Theta(1)$ and the bottom row should only have $\Theta(n)$.

Marking Scheme:
+0.5 per correct table entry

Question 4a

A BST is not always balanced, so BSTs and linked lists have the same worst-case lookup complexity. Depending on insertion order, a key of interest may be easier to access in a linked list than in a BST.

Marking Scheme:
+2 for writing the above solution or a valid variant of it.

Question 4b

Top row should be $\Theta(1)$, bottom row should be $\Theta(b)$

Marking Scheme:
+1 for each entry in the table

Question 5a

```
int **create_array(int num_rows, int num_cols[]){
      int i, **arr;
      arr = calloc(num_rows, sizeof(int*));
      for(i=0; i<num_rows; i++){
            arr[i] = calloc(num_cols[i], sizeof(int));
      }
      return arr;
}
```

Marking Scheme:
+0.5 for allocating memory for int **
+0.5 for looping through each int *
+0.5 for allocating memory for each int *
+0.5 for setting each int to 0

Question 5b

```
void resize_array(int num_rows, int num_cols[], float factors[],
                  int **dyn_arr){
      int i, j;
      for(i = 0; i<num_rows; i++){
            num_cols[i] *= factors[i];
            dyn_arr[i] = realloc(dyn_arr[i], num_cols[i]*sizeof(int));
            for(j=0; j<num_cols[i]; j++){
                  dyn_arr[i][j] = 0;
            }
      }
}
```

Marking Scheme:
+0.5 for updating num_cols[]
+0.5 for looping through each int *
+0.5 for reallocating memory for each int *
+0.5 for setting each int to 0

Question 5c

```
void delete_array(int num_rows, int **dyn_arr){
```

```
      /* num_rows: Number of rows of the sensor array
         dyn_arr: Dynamically sized data structure to deallocate
      */
      for(int i = 0; i < num_rows; i++){
            free(dyn_arr[i]); //OR free(*dyn_arr + i);
      }
      free(dyn_arr);
}
```

Marking Scheme:
0/1: Code didn't make sense and/or missing key elements.
0.5/1: Major syntax/logic errors, incorrect variables/statements used (exception: dyn_arr vs. pp_int), missing free statements, etc. but code at least somewhat makes sense or if only free(dyn_arr) was written.
1/1: Code makes sense, small syntax errors (or none).

Question 5d

q5.h

```
// **Insert code below**
#define NUM_ROWS 40
…
```

q5.c

```
#include <stdlib.h>
// **Insert code below**
#include "q5.h"

int main(void){
      //**Insert code below**
      int **dyn_arr = create_array(NUM_ROWS, num_cols);
      resize_array(NUM_ROWS, num_cols, factors, dyn_arr);
      delete_array(NUM_ROWS, dyn_arr);
      return 0;
}
```

Marking Scheme:
Errors:
- missing statements
- incorrect statements
- statements in the wrong place
- syntax errors
- calling the wrong function or using the wrong inputs/missing inputs

0/2: Code left blank, none of the statements were correct, code didn't have the right idea (if more than 3 lines were written in total), or 4+ errors.

0.5/2: 3 errors, or 4 errors but you had the right idea. Or, at least 1 line correct if there were 3 or less lines written in total (with the exception of a return statement).
1/2: 2 errors, or 3 errors but you had the right idea.
1.5/2: 1 error and you had the right idea.
2/2: perfect.

Note: unnecessary statements were ignored.

Question 5e

int *dyn_arr[] (array of pointers to int) could have been used instead of **int **dyn_arr** (pointer to pointers to int). Recall that, for a pointer ptr, *ptr can equivalently be written as ptr[0], so a pointer can be thought of as pointing to the first element in an array. Whenever an array is passed into a function, the compiler automatically decays this into a pointer, meaning we can choose to pass in an array of pointers to int or a pointer to pointers to int. So passing in int **dyn_arr vs. int *dyn_arr[] into a function results in the exact same functionality.

Marking Scheme:
+0.5 for correctly writing **int *dyn_arr[]**
+0.5 for explaining something about the "equivalence" relationship between pointers and arrays.

Question 6a

```
#include <stdio.h>
void lower_case(char line[]){
     //line: String to convert to lower case
     for(int i = 0; line[i]; i++){
     /* Check condition can be just line[i] or line[i] = '\0'
         Alternatively, can use:
         for(char *l = line; *l; l++)
         Going through entire string: 0.2 pt
         Stopping when reaching '\0': 0.2 pt
     */
          if(line[i] >= 'A' && line[i] <= 'Z')
               line[i] += 'a' – 'A';
          /* Need to check range:
               -0.4 if missing,
               -0.2 if checking in a way that causes errors
               (65 <= line[i] && line[i] <= 90) is also correct
               Need to add the correct value:
               -0.4 if missing,
               -0.2 if adding incorrect value
               Adding 0x40 (64) or doing |= 0x60 is also correct
          */
```

```
        }
        // Option 1
        printf(line);
        // Option 2
        // printf("%s", line);
}
```

Marking Scheme:
1 point for each function, the breakdown is:
+0.5 for iterating over string correctly

- 0 of sizeof(line)
- 0 if iterating from 0 to MAX_STRING without stopping at '\0'

+0.5 for converting correctly

- -0.25 if not checking the case
- -0.15 if checking the case incorrectly

See solution code above for implementation specific breakdowns and remarks. The code for upper_case is the same with a few simple modifications.

Question 6b

```
void write_line(FILE *stream, char file_line[], void (*case)(char
                line[])){
        /*
                output_stream: Stream to write string to
                file_line: String to write to output_stream
                case: Function pointer representing either lower_case or
                        upper_case defined above
        */
        case(file_line);
        // Option 1
        fprintf(stream, "%s\n", file_line);

        // Option 2 (requires printing the "\n" in a separate call
        // fputs(file_line, stream);
        // fputs("\n", stream);

        // Possible incorrect solution since case returns void
        // fprintf(stream, "%s\n", case(file_line));
}
```

Marking Scheme:
1 point in total, the breakdown is:
+0.5 for printing to stream

- Ignored closing the stream at the end (common mistake)
- Ignored missing newline or nonsensically added '\0'
- 0 if opening a new stream
- 0 in some cases when all the arguments were wrong, e.g. passing chars as strings and using *stream

+0.5 for using the function pointer correctly

- 0 if using the return value of the void function case
- 0 in some cases if passing single characters to function

Question 6c

```
#include <stdio.h>
#include <string.h>
#define MAX_STRING 100
int main(void){
      char *file_input = "input.txt";
      char *file_output = "output.txt";

      FILE *in = fopen(file_input, "r");
      FILE *out = fopen(file_output, "w");

      char buff[MAX_STRING];

      int option = 0;
      while(fscanf("%s %d", buff, &option) == 2)
            write_line(out, buff, option ? upper_case : lower_case);
      /* Marks for reading and parsing depnd on the stop condition,
         format string,
         and syntax
      */

      fclose(in);
      fclose(out);

      return 0;
}
```

Marking Scheme:

3 points in total, the breakdown is:

+0.6 for opening streams

- Must use fopen()
- Valid modes for input stream are "r", "r+", "a+"
- Valid modes for output stream are "w', "w+", "a", "a+"
- -0.1 for each incorrect syntax

- -0.2 for each incorrect mode

+1.2 for reading and parsing stream

+0.8 for writing to stream

- Must use write_line()
- +0.3 for passing valid arguments
- +0.5 for using function pointers correctly

+0.4 for closing streams