ENEE 140, Fall 2023
Midterm Exam — Answer Key
Do Not Make a Copy!!

Date:

University of Maryland Honor Pledge: The University is committed to Academic Integrity, and has a nationally recognized Honor Code, administered by the Student Honor Council. In an effort to affirm a community of trust, the Student Honor Council proposed and the University Senate approved an Honor Pledge. The University of Maryland Honor Pledge Reads:

"*I pledge on my honor that I have not given or received any unauthorized assistance on this examination (or assignment)*"

Please write the exact wording of the Pledge, followed by your signature, in the space below:

Pledge: ______________________________
Pledge: ______________________________
Pledge: ______________________________
Pledge: ______________________________

Your signature: ______________________________
Full name: ____________________ Course: ________ Directory ID: ____________

List of Exam Questions:

Question:	1	2	3	4	5	6	7	Total
Points:	4	9	22	15	14	16	20	100
Score:								

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name, your section and your Directory ID on the front.
- Write your name and section at the bottom of each page as well.

- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. No electronic devices (e.g. laptops, tablets, smartphones, calculators) are allowed. Good luck!

1. (4 points) This problem tests your understanding of integer arithmetic. Assume operations take place on the grace machine (w = 32).

```c
int a = ((UINT_MAX-1) / 2) + 1;
```

What is the signed value of `a`?

A. `0`

B. `1`

C. `INT_MIN`

D. `INT_MAX`

1. **C**

Solution:

$$\begin{aligned} \texttt{((UINT\_MAX - 1)/ 2)+ 1} &= ((2^{32} - 2)/2) + 1 \\ &= (2^{31} - 1) + 1 \\ &= \texttt{INT\_MAX} + 1 \\ &= \texttt{INT\_MIN} \end{aligned}$$

2. (9 points) This problem tests your understanding of functions and arithmetic conversions. What is the output of the following code?

```c
#include <stdio.h>

int fun1(int a, int b);
float fun2(int a);
char fun3(float a);

int main() {
    int a = fun1(7, 5);
    float b = fun2(a);
    char c = fun3(b);

    printf("%d\n", a); /*Answer 1*/
    printf("%.3f\n", b); /*Answer 2*/
    printf("%c\n", c); /*Answer 3*/

    return 0;
}

int fun1(int a, int b) {
    int sum = a + b;
    sum += 3;
    return sum;
}

float fun2(int a) {
    float result = (a * 2) / 4.0;
    result -= 5;
    return result;
}

char fun3(float a) {
    int b = a/3.0;
    if(b < 0.5)
        return 'A';
    else
        return 'B';
}
```

(a) Answer 1

(b) Answer 2

(c) Answer 3

(a) **15**

(b) **2.500**

(c) **A**

Solution: Answer 1 = 15, Answer 2 = 2.500, Answer 3 = A

3. (22 points) This problem tests your understanding of `while` and `for` loops, as well as `if-else` branches.

   (a) What does the following code output?

```
#include <stdio.h>

int main() {
    int i = 0;

    while (i < 10) {
        if (i == 6) {
            printf("Humongous ");
            i = i - 5;
        } else if (i > 6) {
            i++;
        } else {
            printf("Big ");
            i = i + 3;
        }
        printf("Hamburger.\n");
    }
    return 0;
}
```

Solution:

```
Big Hamburger.
Big Hamburger.
Humongous Hamburger.
Big Hamburger.
Big Hamburger.
Hamburger.
Hamburger.
Hamburger.
```

   (b) If you could change only one line in the program, could this code be converted to use a `for` loop instead of a `while` loop?

   A. `True`

   B. `False`

(b) **A**

   (c) Can ALL `while` loops be converted to `for` loops?

A. `True`

B. `False`

(c) **A**

(d) Can ALL `for` loops be converted to `while` loops?

A. `True`

B. `False`

(d) **A**

4. (15 points) This question tests your ability to debug code. The following code takes a user input of two integers, inputs the integers into the sum function and assigns the value of the function to output. Output is then printed out with a width of 5. Circle the line numbers that contain bugs, and in the space to the right of the code explain why it is a bug. (Hint: there are bugs on 5 lines)

```
1  #Include <stdio.h>
2
3  int sum(a, b);
4
5  int main(){
6      int x, y, output;
7      printf("Enter two numbers");
8      scanf("%d%d", x, y);
9
10     sum(x, y) = output;
11
12     printf("%5.1d", output);
13
14     return 0;
15 }
16
17 int sum(int a, int b){
18     return a + b;
19 }
```

Solution:

`Line 1: #Include should be #include`

`Line 3: Need to specify types of parameters`

`Line 8: scanf needs to have its variables have a &x and &y`

```
Line 10: cannot initialize a function to a variable

Line 12: Cannot use 5.1 as a format specifier for %d
```

The correct code is:

```
1  #include <stdio.h>
2
3  int sum(int a, int b);
4
5  int main(){
6      int x, y, output;
7      printf("Enter two numbers");
8      scanf("%d%d", &x, &y);
9
10     output = sum(x, y);
11
12     printf("%5d", output);
13
14     return 0;
15 }
16
17 int sum(int a, int b){
18     return a + b;
19 }
```

5. (14 points) This question tests your knowledge of functions and for loops.

A prime number is an integer N whose only factors are 1 and N. For example, 31 is a prime number because its only factors are 1 and 31 (1 * 31 = 31). 2, 3, 5, 7, 11, 13, 17 are also prime numbers for the same reason.

The `is_prime_number` function takes in a positive integer `n` (greater than 1) and uses division in a for loop to check all of its potential factors up to the square root of n. If there is a number `i` (not equal to 1 or `n`) such that the remainder of `(n / i)` equals 0, the function returns 0, indicating that the number `n` is not a prime number. Otherwise, it returns 1. Fill in the blanks to make `is_prime_number` work properly.

The code below uses the `sqrt()` function from the `math.h` library to get the square root of a number.

```
int is_prime_number(______int______ n) {

  int i;

  for (_____i = 2_____;_______i_______ <= sqrt(n);______i++______) {

    if ((______n % i_____)==0) {
      return ________0________;
    }
  }

  return ________1________;
}
```

Solution:

```
int is_prime_number(int n) {
  int i;

  for (i = 2; i <= sqrt(n); i++) {
    if ((n % i) == 0) {
      return 0;
    }
  }
  return 1;
}
```

6. (16 points) This problem tests your understanding of using state variables in `getchar` user-input parsing.

   The year is 1961 and you are stationed undercover in East Berlin. You are given an urgent order from the CIA director to listen for an abort signal on your transceiver. The signal will be sent as a jumbled string of characters, ending with the `EOF` character. The abort code will be signified if the transmission contains exactly 5 occurrences of **SOS**. The transceiver will decode the signal through user-input.

   The purpose of the following code is to count the number of times the character sequence **SOS** occurs. Your job is to complete the code so you can stay out of the gulag.

```
#include <stdio.h>

#define NEXT_STATE state++;
#define RESET_STATE state = 0;

int main() {
    int cur;
    int state = ____0____; /* Keeps track of the place in the sequence */
    int count = 0;         /* Counter for the sequence occurrence */

    while ((cur = getchar()) != EOF) {
        if (state == 0) {
            if (cur == 'S') {
                NEXT_STATE
            }
        } else if (state == 1) {
            if (cur == 'O') {
                NEXT_STATE
            } else {
                RESET_STATE
            }
        } else if (state == 2) {
            if (cur == 'S') {
                ____count++____;
                RESET_STATE
            } else {
                RESET_STATE
            }
        }
    }

    printf("The sequence \"SOS\" occurs %d times in the signal.\n", ____count____);

    return 0;
}
```

Solution:

```c
#include <stdio.h>

#define NEXT_STATE state++;
#define RESET_STATE state = 0;

int main() {
    int cur;
    int state = 0; /* State variable to keep track of the place in the sequence */
    int count = 0; /* Counter for the sequence occurrence */

    while ((cur = getchar()) != EOF) {
        if (state == 0) {
            if (cur == 'S') {
                NEXT_STATE
            }
        } else if (state == 1) {
            if (cur == 'O') {
                NEXT_STATE
            } else {
                RESET_STATE
            }
        } else if (state == 2) {
            if (cur == 'S') {
                count++; /* Sequence found, increment the count */
                RESET_STATE
            } else {
                RESET_STATE
            }
        }
    }

    printf("The sequence \"SOS\" occurs %d times in the signal.\n", count);

    return 0;
}
```

7. (20 points) This problem will test your understanding of character manipulation and use of constants with control flow. The intent of this program is to act like a file-naming program. It should take in one newline-terminated line of user input. Its output will be the same as the input, except for the following modifications:
(1) Convert every *sequence* of non-alphanumeric characters to a single underscore (looks like '_').
(2) If the first character after an underscore is a letter, capitalize it.

It should do this with the following limitations:
(1) Do not use the `ctype.h` library.
(2) Print no more than one underscore in a row.

For example, an entry of `'abc aBC123   123abc'` should print out `'Abc_ABC123_123abc'`. Fill in the blanks within `main()` to make this occur.

```
#include <stdio.h>
#define OUT_OF_WORD 0
#define IN_WORD     1

int main()
{

    int c, state = OUT_OF_WORD;

    while((c = getchar()) != ______'\n'______) {
        if ((____c >= 'a'____ && c <= 'z') ||
            (____c >= 'A'____ && ____c <= 'Z'____) ||
            (____c >= '0'____ && ____c <= '9'____)) {
            if (state == OUT_OF_WORD) {
                if (c >= 'a' && c <= 'z')
                    printf("%c", _c + 'A' - 'a'_);
                else
                    printf("%c", ________c________);
                state = IN_WORD;
            }
            else {
                printf("%c", c);
                state = _____IN_WORD_____;
            }
        }
        else {
            if (state == _____IN_WORD_____)
                printf("_");
            state = OUT_OF_WORD;
        }
    }
    return 0;
}
```

Solution:

```
#include <stdio.h>
#define OUT_OF_WORD 0
#define IN_WORD   1

int func(char x);


int main(){

int c, state = OUT_OF_WORD;

    while((c = getchar()) != '\n')
    {
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c <=
             '9'))
        {
            if (state == OUT_OF_WORD)
            {
                if (c >= 'a' && c <= 'z')
                    printf("%c", c + 'A' - 'a');
                else
                    printf("%c", c);
                state = IN_WORD;
            }
            else
            {
                printf("%c", c);
                state = IN_WORD;
            }
        }
        else
        {
            if (state == IN_WORD)
                printf("_");
            state = OUT_OF_WORD;
        }
    }
    return 0;
}
```