Student Name: ________________ NetID: ________________ RUID: ________________

Midterm Exam
Monday, March 20, 2023
12:10 pm – 1:30 pm
Hill 114

Rutgers University
01:198:211 Computer Architecture Sections 5-8
Spring 2023
Instructor: Prof. Yipeng Huang

Instructions

- Please turn off all electronic devices you have in possession.
- This exam consists of 8 pages. Check that you have all eight pages right away.
- Write your name, NetID, and RUID student number at top right of this page.
- Write your name on the separate bubble sheet and bubble in your 9-digit RUID number under Student Number, using columns 0 through 8. Ignore the 9th column.

- This exam consists of 40 questions, all are single-selection multiple choice out of 5 choices.
- You will receive +1.0 point for each correct answer, -0.25 points for each incorrect answer.
- Mark your answers on the bubble sheet which has room for 50 questions. Use only the first 40. Ignore the remaining 10.
- The problems are not sorted by difficulty. Skip ahead to work on sections you find easier.

- This is a closed book, closed notes exam. No electronic devices are permitted. Accessing any prohibited materials during the exam will lead to a score of 0 on this exam.
- We may check your RUID card when collecting the exam booklet and the bubble sheet.

C Programming

The following set of 10 questions are about the C language syntax, arrays, pointers, managing memory, and potential bugs. You can assume the programs are compiled on a computer like iLab, using the following configuration of the gcc compiler (typical for the programming assignments in this class):

```
gcc -Wall -Werror -fsanitize=address -std=c99 -o midterm midterm.c -lm
```

You are asked what will be printed to the command line. Select the correct answer among five options.

1.

```
#include <stdio.h>

int main() {
    int x, y;
    int* p = &x;
    int *q = &y;
    *p = 2;
    *q = *p + 3;
    printf("%d\n", y);
    return 0;
}
```

A. 2
B. 3
C. 4
D. 5
E. 6

2.

```
#include <stdio.h>

void fun (int** ptr) {
    **ptr = **ptr * **ptr;
}

int main() {
  int y = 3;
  int* pointer = &y;
  fun ( &pointer );
  printf("%d\n", y);
  return 0;
}
```

A. 3
B. 9
C. 27
D. ERROR: AddressSanitizer: stack-buffer-overflow
E. ERROR: LeakSanitizer: detected memory leaks

3.

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

int* function( int* pointer ) {
    pointer = malloc ( sizeof(int) );
    pointer[0] = 1;
    return pointer;
}

int main() {
    int* array = NULL;
    array = function( array );
    printf ( "%d\n", array[0] );
    free(array);
    return 0;
}
```

A. -1
B. 0
C. 1
D. 1, followed by ERROR: LeakSanitizer: detected memory leaks
E. ERROR: AddressSanitizer: heap-buffer-overflow

4.

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

void function( int* pointer ) {
    printf( "%d\n", (*pointer)+2 );
}

int main() {
    int* array = calloc( 2, sizeof(int) );
    array[0] = 1;
    array[1] = 0;
    function( array );
    free ( array );
    return 0;
}
```

A. 0
B. 1
C. 3
D. ERROR: AddressSanitizer: heap-buffer-overflow
E. ERROR: AddressSanitizer: attempting double-free

5.

```
#include <stdio.h>

int main() {
    int x[1] = {10};
    printf( "%d\n", *(x+1) );
    return 0;
}
```

A. 9
B. 10
C. 11
D. ERROR: AddressSanitizer: stack-buffer-overflow
E. ERROR: AddressSanitizer: heap-buffer-overflow

6.

```
#include <stdio.h>

int main() {
    double realNum = 900;
    double* pointer = &realNum;
    double test = **&pointer;
    printf("%lf\n", test);
    return 0;
}
```

A. Compile time syntax error
B. 900
C. 900.000000
D. A hexadecimal address value
E. ERROR: AddressSanitizer: SEGV on unknown address

7.

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

typedef struct quiz {
    int data;
    struct quiz* next;
} quiz_t;

int main () {
    quiz_t* myQuizType =
malloc(2*sizeof(quiz_t));
    (*myQuizType).data = 1;
    (*myQuizType).next = myQuizType+1;
    myQuizType->next->data = 2;
    printf("%d\n", myQuizType[1].data);
    free (myQuizType);
    return 0;
}
```

A. 0
B. 1
C. 2
D. ERROR: AddressSanitizer: SEGV on unknown address
E. ERROR: AddressSanitizer: heap-buffer-overflow

8.

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

int main () {
    int* pointer0 = malloc(sizeof(int));
    *pointer0 = 2;
    int* pointer1 = pointer0;
    *pointer1 = 1;
    printf("%d\n", *pointer0);
    free(pointer0);
    free(pointer1);
```

A. 0
B. 1
C. 2
D. 1, followed by ERROR: AddressSanitizer: attempting double-free
E. ERROR: AddressSanitizer: heap-use-after-free

```
        return 0;
    }
```

9.

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

    int quizSwap ( int* a, int b ) {
        int temp = *a;
        *a = b;
        b = temp;
        return b;
    }

    int main () {
        int x = 2;
        int y = 3;
        int z = quizSwap ( &x, y );
        printf ( "%d\n", x+y+z );
    }
```

A. 5
B. 6
C. 7
D. 8
E. 9

10.

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

    int* funcReturn ( int* a ) {
        *a = 5;
        return a;
    }

    int main () {
        int val = 7;
        printf ( "%d\n", *funcReturn(&val) );
    }
```

A. Compile time syntax error
B. 0
C. 5
D. 7
E. A hexadecimal address value

Integers

The following set of 10 questions are about the data representation of bits, bytes, integers, and operations. You can assume the programs are compiled the same way as the last section. You are asked what will be printed to the command line. Select the correct answer among five options.

11.

```
    #include <stdio.h>

    int main() {
        signed char number = 127;
        number = ~number;
        printf("%d\n", number);
        return 0;
    }
```

A. -128
B. -127
C. 1
D. 127
E. 128

12.

```
#include <stdio.h>

int main() {
    signed short number = 1;
    number = number<<9;
    printf("%d\n", number);
    return 0;
}
```

A. 0
B. 2
C. 9
D. 512
E. 1024

13.

```
#include <stdio.h>

int main() {
    signed char number = 3;
    number = number<<7;
    printf("%d\n", number);
    return 0;
}
```

A. -384
B. -128
C. 21
D. 128
E. 384

14.

```
#include <stdio.h>

int main() {
    signed char number = -128;
    number--;
    printf("%d\n", number);
    return 0;
}
```

A. -129
B. -127
C. 0
D. 1
E. 127

15.

```
#include <stdio.h>

int main() {
    signed char number = -64;
    number = number>>5;
    printf("%d\n", number);
    return 0;
}
```

A. -4
B. -2
C. -1
D. 2
E. 6

16.

```
#include <stdio.h>

int main() {
    unsigned char number = 192;
    number = number>>5;
    printf("%d\n", number);
    return 0;
}
```

A. -4
B. -2
C. -1
D. 2
E. 6

17.

```
#include <stdio.h>

int main() {
    printf("%d\n", 9 & 5);
    return 0;
```

A. 1
B. 5
C. 9
D. 12
E. 13

```
    }
```

18.

```
    #include <stdio.h>

    int main() {
        printf("%d\n", 9 | 5);
        return 0;
    }
```

A. 1
B. 5
C. 9
D. 12
E. 13

19.

```
    #include <stdio.h>

    int main() {
        printf("%d\n", 9 ^ 5);
        return 0;
    }
```

A. 1
B. 5
C. 9
D. 12
E. 13

20.

```
    #include <stdio.h>

    int main() {
        printf("%d\n", 0x11 + 010);
        return 0;
    }
```

A. 19
B. 21
C. 25
D. 27
E. 33

Floating point numbers

For the following 15 questions, we will explore the properties of a 16-bit half precision floating point format. The 16 bits are used in the encoding as follows:

- The most significant (leftmost) bit encodes s, the sign.
- The next k=5 bits encode the exponent. The exponent bias is $2^{k-1} - 1 = 15$.
- The remaining 10 bits are the frac bits, which encode the mantissa.

The rules are the same as the IEEE 754 standard for 32-bit and 64-bit floating point numbers.

The following 5 questions ask you to match each floating point numerical value to its binary representation, written as a sequence of 0s and 1s, or as a hexadecimal number. Select the right representation from a shared set of five options.

21. 0.0	A. 0_11111_0000000000 B. 0x8000 C. 1_11111_0000000001 D. 0_00000_0000000000 E. 0xFC00
22. -0.0	
23. +inf	
24. -inf	
25. NaN, not a number	

The following 5 questions ask you to match each special value in this half precision floating point number system to its binary representation, written as a sequence of 0s and 1s. Select the right representation from a shared set of five options.

26.	Largest magnitude representable negative number	A. 1_00000_0000000001
27.	Negative one	B. 1_00000_1111111111
28.	Smallest magnitude normalized negative number	C. 1_00001_0000000000
29.	Largest magnitude denormalized negative number	D. 1_01111_0000000000
30.	Smallest magnitude negative number	E. 1_11110_1111111111

The following 5 questions ask you to match each binary representation (some written as hexadecimal numbers) of a half precision floating point number to its corresponding real number numerical value. Select the right value from a shared set of five options.

31.	`1_00001_0000000000`	A. -768.0
32.	`0x0100`	B. -1.0
33.	`1_01111_0000000000`	C. -1./16384
34.	`0x7A00`	D. 1./65536
35.	`1_11000_1000000000`	E. 49152.0

The following set of 5 questions are about properties of the floating point numbers. You can assume the programs are compiled the same way as the first two sections. You are asked what will be printed to the command line. Select the correct answer among five options.

36.

```
#include <stdio.h>

int main() {
    printf("%f\n", (float) (-4/3));
    return 0;
}
```

A. Compile time syntax error
B. -2.000000
C. -1.333333
D. -1.000000
E. -1

37.

```
#include <stdio.h>

int main() {
    printf("%f\n", (float) (-4./3));
    return 0;
}
```

A. Compile time syntax error
B. -2.000000
C. -1.333333
D. -1.000000
E. -1

38.

```
#include <stdio.h>

int main() {
  double bigPos = 1e10;
  printf("%lf\n", 1.0 + (bigPos - bigPos));
  return 0;
}
```

A. 0.000000
B. 1.000000
C. 10000000000.000000
D. inf
E. NaN

39.

```c
#include <stdio.h>
#include <float.h>

int main() {
    float x = (float) DBL_MAX;
    float y = FLT_MAX;
    printf("%f\n", x-y);
    return 0;
}
```

A. -inf
B. 0.000000
C. FLT_MAX
D. inf
E. NaN

40.

```c
#include <stdio.h>

int main() {
    printf("%f\n", -1/0.);
    return 0;
}
```

A. Compile time syntax error
B. -inf
C. -1.000000
D. 0.000000
E. NaN

This concludes the exam.