Csci 201: Sample Exam

Duration: 65 minutes

Name:_____

NetID:_____

Student to your left: **Student to your right:**

_____ _____

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED

Instructions:

- **Write your full name and your NetID on the front of this exam.**
- Make sure that your exam is not missing any sheets. There should be seven (6) double sided pages in the exam.
- Write your answers in the space provided below each problem. If you make a mess, clearly indicate your final answer.
- **Answer all questions in this exam.**
- If you have any questions during the exam, raise your hand and we will try to get to you.
- At the end of the exam, there are blank pages. Use them as your scrap paper. If you need additional scrap paper, raise your hand and we will get it for you.
- This exam is closed books, closed notes, closed computers. You are allowed to have a single double sided sheet of paper with anything you wish on it. That sheet should have your name on front an back.
- **You need to stay in your seat until the exam is finished.** You should not leave the room even if you finish the exam. This distracts other students who are still working.

Good luck!

1 (40 points):	
2 (12 points):	
3 (8 points):	
4 (14 points):	
5 (14 points):	
6 (12 points):	
TOTAL (100):	

Joanna Klukowska
joannakl@cs.nyu.edu

Problem 1 (40 points) Multiple Choice Questions.

A. (4 points) What does the following declaration mean?

```
int* ptr [10];
```

(a) `ptr` is an array of pointers to 10 integers

(b) `ptr` is a pointer to an array of 10 integers

(c) `ptr` is an array of 10 integers

(d) `ptr` is a pointer to an array

B. (4 points) What is the output of this program?

```
#include<stdio.h>
void fun(int*, int*);
int main()
{
    int i=5, j=2;
    fun(&i, &j);
    printf("%d, %d", i, j);
    return 0;
}
void fun(int *i, int *j)
{
    *i = *i + *j;
    *j = *i + *j;
}
```

(a) 5, 2

(b) 7, 7

(c) 7, 9

(d) 9, 9

(e) none of the above

C. (4 points) Convert the floating point `10.001` in binary into its equivalent decimal notation (note, this is NOT IEEE 754 notation).

(a) 2.25

(b) 2.001

(c) 10.125

(d) 2.125

D. (4 points) What is the value of the following bit-vector `11001` when it is interpretted as an integer represented in two's complement notation.

(a) -6

(b) 6

(c) -7

(d) 7

(e) 25

(f) -25

(g) none of the above

E. (4 points) What is the value of the following bit-vector `11001` when it is interpretted as an integer represented in unsigned notation.

(a) -6

(b) 6

(c) -7

(d) 7

(e) 25

(f) -25

(g) none of the above

Joanna Klukowska
joannakl@cs.nyu.edu

F. (4 points) Assuming that the integer is 2 bytes and we use the two's complements notation, what will be the output of this program? Assume arithmetic shift.

```
#include<stdio.h>

int main()
{
  printf("%x\n", -1 >> 1);
  return 0;
}
```

(a) `ffff`
(b) `0000`
(c) `0fff`
(d) `fff0`
(e) `0001`
(f) `8000`
(g) none of the above

G. (4 points) Assuming that the integer is 2 bytes and we use the two's complements notation, what iwill be this output of the program?

```
#include<stdio.h>

int main()
{
  unsigned int a=0xffff;
  a= ~a;
  printf("%x\n", a);
  return 0;
}
```

(a) `ffff`
(b) `0`
(c) `-1`
(d) `fffe`
(e) none of the above

H. What happens when the following function is called? You can assume that the `nums` array contains at least `size` many elements.

```
void function ( int nums [], unsigned int size ) {
  int i = 0;
  for ( i = 0; i < size; i++ ) {
    printf ( "%d, ", nums[i] ) ;
  }
}
```

(a) It cannot be called because the code does not compile.
(b) It executes, the `size` many values in the `nums` array are printed and then the function terminates.
(c) It executes, the `size` many values in the `nums` array are printed and then the function prints those same elements again and again and ... in an infinite loop.
(d) It executes, the `size` many values in the `nums` array are printed and then the function attempts to print content of other memory locations. This may result in the program crashing.

I. (8 points) Circle all statements below that are **true**.

(a) At most one operand of an x86-64 assembly instruction can be a register.
(b) C does not have a boolean type; integer values are used instead.
(c) In C, a formal parameter that is an array of strings could have the type `char **`.
(d) Consider a **5-bit** two's complement representation. The largest positive number that can be represented in this notation is 32.
(e) Assume that the register `%rdi` stores a value of 3 and then the following instructions are executed:
```
leaq (%rdi, %rdi, 2), %rax
salq $2, %rax
```
The value stored in register
(f) All object files can be executed in a machine on which they were created.
(g) A special register, called condition code, is set to 1 whenever an overlow occurs.
(h) x86-64 has different instructions for different kinds of loops that correspond to the C's loops.

Joanna Klukowska
joannakl@cs.nyu.edu

Problem 2 (12 points) .

Consider the following C functions and assembly code:

```
1  int fun4(int *ap, int *bp)
2  {
3      int a = *ap;
4      int b = *bp;
5      return a+b;
6  }
7
8
9  int fun5(int *ap, int *bp)
10 {
11     int b = *bp;
12     *bp += *ap;
13     return b;
14 }
15
16 int fun6(int *ap, int *bp)
17 {
18     int a = *ap;
19     *bp += *ap;
20     return a;
21 }
```

```
1   pushl %ebp
2   movl %esp,%ebp
3   movl 8(%ebp),%edx
4   movl 12(%ebp),%eax
5   movl %ebp,%esp
6   movl (%edx),%edx
7   addl %edx,(%eax)
8   movl %edx,%eax
9   popl %ebp
10  ret
```

Which of the functions compiled into the assembly code shown?

Joanna Klukowska
joannakl@cs.nyu.edu

Problem 3 (8 points) .

Assume we are executing the following program on a 64-bit machine (assume that the system uses 8 bytes to store a pointer, 4 bytes to store a variable of type `int` and 8 bytes to store a variable of type `double`):

1. What is the output?
2. If we change `p`'s declaration to be `double *p;` and use the type casting with `malloc` to be `(double*)`, what is the output?
3. In both cases, how many bytes are freed by calling `free(...)`?

```
1 int main() {
2   int *p;
3   p = (int *) malloc( 128 );
4   printf("%d\n", sizeof(p));
5   free(p);
6   return 0;
7 }
```

Joanna Klukowska
joannakl@cs.nyu.edu

Problem 4 (14 points) .

Write a function that given a pointer to the root of a binary search tree (possibly equal to NULL for an empty tree) and a string, adds a new node in the appropriate position of the BST.
The definition of the node is as follows

```
struct bst_node {
  char * word;
  struct bst_node * left;
  struct bst_node * right;
};
```

Joanna Klukowska
joannakl@cs.nyu.edu

Problem 5 (14 points) .

Consider the following 5-bit floating point representation based on the IEEE floating point format. There is a sign bit in the most significant bit. The next three bits are the exponent, with an exponent bias of 3. The last bit is the fraction. The rules are like those in the IEEE standard (normalized, denormalized, representation of 0, infinity, and NaN).

We consider the floating point format to encode numbers in a form:

$$V \;=\; (-1)^s \times M \times 2^E$$

where M is the significand and E is the exponent.

Fill in missing entries in the table below with the following instructions for each column:

Description: Some **unique** property of this number, such as, "The largest denormalized value."

Binary: The 5 bit representation.

M**:** The value of the significand/mantissa written in decimal format.

E**:** The integer value of the exponent written in decimal format.

Value: The numeric value represented by the number, written in decimal format.

You do not need to fill in entries marked "—". For the arithmetic expressions, recall that the rule with IEEE format is to round to the number nearest the exact result. Use "round-to-even" rounding.

Description	Binary	M	E	Value
Minus Zero				-0.0
Positive Infinity		—	—	$+\infty$
	`01101`			2
Smallest number > 0 (small positive fraction)				
—				1.0

Use this space for your work, but enter all the answers into the table.

Joanna Klukowska
joannakl@cs.nyu.edu

Problem 6 (12 points) .

Consider the following `inplace_swap` function:

```
1 void inplace_swap (int *x, int *y) {
2     *y = *x ^ *y; /* step 1 */
3     *x = *x ^ *y; /* step 2 */
4     *y = *x ^ *y; /* step 3 */
5 }
```

Starting with the values of `a` and `b` in the locations pointed to by `x` and `y` (these values are indicated in thef first row of the table), fill in the table that follows giving the values stored at the two locations **after** each step of the function.

Step	*x or a	*y or b
Initially	0100 1001	0011 1011
Step 1		
Step 2		
Step 3		

Use this space for your work, but enter all the answers into the table.

Joanna Klukowska
joannakl@cs.nyu.edu

SCRAP PAPER NAME _____

Joanna Klukowska
joannakl@cs.nyu.edu

SCRAP PAPER NAME _____

Joanna Klukowska
joannakl@cs.nyu.edu

SCRAP PAPER NAME _____