CS 2130 Final Exam

Name _____

You MUST clearly write your computing ID on **EACH** page and put your name on the top of this page, too. Please write legibly.

Write your answers in the box labeled "Answer" when provided. In any multiple choice answer, **fill in the circle completely** for credit; checkmarks, circles, lines, or other marks will be graded as an empty circle.

If you are still writing when "pens down" is called, your exam will not be graded – even if you are still signing the honor pledge. So please do that first. Sorry to have to be strict on this!

There are 16 pages to this exam. Once the exam starts, please make sure you have all the pages. Questions are worth different amounts of points, so be sure to look over all the questions and plan your time accordingly.

This exam is CLOSED text book, closed-notes, closed-cell phone, closed-smart watch, closed-computer, closed-neighbor, etc. You may **not** discuss this exam with anyone until after the grades have been released. Please sign the honor pledge below.

On my honor as a student, I have neither given nor received aid on this exam. I will not discuss the content of this exam, even in vague terms, with *anyone* other than current course staff, until *after* grades have been released.

_____

The Force is strong with this one.

–Darth Vader

Page 2: Toy ISA to C

Throughout the semester, we took a bottom-up approach, building from electricity on wires to C. The next few questions take a top down approach to Computer Systems and Organization.

Consider the following recursive C function to calculate the "funfact" of two numbers; for example, $\texttt{funfact}(5,2) = 2 + 10 + \texttt{funfact}(4,2) = 12 + 2 + 8 + \texttt{funfact}(3,2) = ... = 41$.

```
long funfact(long n, long e) {
    if (n < 1)
        return 1;
    return e + (2 * n) + funfact(n-1, e);
}
```

1. [10 points] Rearrange the x86-64 assembly instructions to implement `funfact`. For each section, reorder the instructions by writing the number of the appropriate instruction on the lines provided to the right. Some order has been provided for you and the order in one section *may* influence the ordering in another section. *Each instruction may be used at most once and only within its group. Some instructions may not be used. All blanks will be filled.*

Possible Instructions	**Proper Ordering**
1. `cmpq $0x0, %rdi`	2 `funfact:`
2. `funfact:`	____
3. `jb label1`	____
4. `jle label1`	____
5. `leaq 2(%rsi, %rdi), %rbx`	____
6. `leaq (%rsi, %rdi, 2), %rbx`	____
7. `pushq %rbx`	
8. `addq $-1, %rdi`	____
9. `addq %rbx, %rax`	____
10. `callq funfact`	____
11. `jmp funfact`	14 `popq %rbx`
12. `label1:`	
13. `movl $1, %eax`	____
14. `popq %rbx`	____
15. `retq`	____
16. `retq`	16 `retq`
17. `subq $-1, %rdi`	

Page 3: Toy ISA to C, continued

2. [9 points] Implement `funfact` using our Toy ISA and following our calling conventions. Part of the code has been provided for you; fill in the missing instructions. In this implementation, assume the function is loaded at memory address `0x20`. *Our Toy ISA is described on page 15*.

```
// Check for base case (n < 1)
r1 = 0x2E
```

```
// Calculate return value (with recursive call)
```

```
// Handle base case
r0 = 0x01
pop r1
return
```

Write your program in binary below. Only the final program below will be graded for this question. If you use fewer instructions, use no-ops to fill in the rest. We have filled in some of the instructions and immediate values already.

	64	2E										
20	21	22	23	24	25	26	27	28	29	2a	2b	2c

	60	01	85	83	00	00	00	00	00	00	00	00
2d	2e	2f	30	31	32	33	34	35	36	37	38	39

Page 4: Binary

3. [3 points] Convert the decimal value -42 into 8-bit two's complement binary. Answer in hexadecimal.

Answer

4. [3 points] Provide a value of x for which $y = x + \texttt{0xF9}$ sets y to 0. Assume x and y are both an `unsigned char`. Answer in hexadecimal.

Answer

5. [4 points] Given the following array of `unsigned short` values, store them into 4 bytes of memory under each of the following assumptions.

```
unsigned short x[2] = {0x0268, 0x4573};
```

A. Assume little-endian storage.

B. Assume big-endian storage.

6. [3 points] Which of the following C expressions are equivalent to `x ^ y`, where `x` and `y` are declared as ints? (Fill in the circle completely for all that apply.)

- ◯ `~(~x & ~y)`
- ◯ `(x & ~y) | (~x & y)`
- ◯ `(x | ~y) & (~x | y)`
- ◯ `!(~x & ~y)`
- ◯ `~x | ~y`
- ◯ `~(x + y)`

Page 5: Circuits

7. [9 points] In the space below, draw a circuit with three 1-bit inputs **x**, **y**, and **z** that outputs the result **x + y + 2z**. Label your outputs as **a**, **b**, ... (as many as you need to avoid overflow), with **a** as the low-order bit, **b** as the next lowest-order bit, and so on, up to the highest-order bit of the output value. Construct the circuit using only wires and gates from the set {and, or, not, xor}. Clearly label your input and output wires. *Hint: writing out a truth table can help design and/or check your circuit.*

Page 6: C

8. [3 points] Given the following function macro, what is the value of `x` below?

```
#define FIVEFOUR(y) 5*y + 4
// ...
int x = 2 * FIVEFOUR(1 + 2);
```

◯ 16

◯ 22

◯ 34

◯ 38

9. [4 points] Answer the following True/False questions by writing either `T` or `F` in the blank provided.

____ Short function definitions, such as `square` below, should be included in header files.
`int square(int x) { return x*x; }`

____ The `syscall` instruction always calls the same function in kernel memory.

____ Garbage collectors, such as those in Java, automatically free unreachable memory.

____ Garbage is a subset of unreachable memory.

10. [3 points] As in homework 10, suppose we wrote a chat program that takes two command line arguments: an ip address and a port number. That is, we would run our program as `./a.out 127.0.0.1 54321`. Which of the following would print the second argument to standard out?

◯ `printf("%d", argv[2]);`

◯ `printf("%s", argv[2]);`

◯ `printf("%d", argv[1]);`

◯ `printf("%s", argv[1]);`

Page 7: Writing C

11. [14 points] Implement the `csostrncmp` function, included in the manual page on this exam. You may **not** use any library functions. *Hint: strings may be less than n characters.*

```
int csostrncmp(const char *s1, const char *s2, size_t n) {
```

```
}
```

Page 8: Memory

Consider the following code, shown with line numbers which are not part of the code itself. The code contains one or more memory errors. Use this code for the next 3 questions.

```
 1 int checkLine(char *line) {
 2     char *prev;
 3     while(line) {
 4         char *t = strsep(&line, " ");
 5         if (prev && prev[0] == t[0]) {
 6             return 1;
 7         }
 8         prev = t;
 9     }
10     return 0;
11 }
12
13 void findRepeats() {
14     int length = 4096;
15     char *buffer = (char *) malloc(sizeof(length));
16     while (fgets(buffer, length, stdin) != NULL) {
17         if (checkLine(buffer))
18             puts("Repeated first letter");
19         else
20             puts("Unique words");
21     }
22 }
```

12. [3 points] The code above has one memory leak. After which line should we add a `free`? For example, if we should free before the `return 0;`, write **9**.

Answer

13. [3 points] What should be freed? That is, what should we pass to the call `free();`? If the pointer to be freed is no longer available at that point, write an X in the box instead.

Answer

Page 9: Memory and C

14. [5 points] For each of the following memory errors, give the line where the error occurs in the code on the previous page. If the error does not occur, put an X in on the line.

    ____ Accesses uninitialized memory

    ____ Escaping pointer

    ____ Fails to use `sizeof` or uses `sizeof` incorrectly

    ____ Possible heap buffer overflow

    ____ Possible stack buffer overflow

15. [4 points] Consider the following C code. Which of the following could be used to fill in the blank to access the `score` element of each player in the list? (Fill in the circle completely for all that apply.)

```
typedef struct { char *name; double score; int rank; } player;

double calculateAvgScore(player *list, size_t n) {
    double average = 0.0;
    for (int i = 0; i < n; i++) {
        average += _________________________;
    }
    return average / n;
}
```

◯ `(list[i]).score`

◯ `(*(list + i)).score`

◯ `(list[i])->score`

◯ `(list + i)->score`

◯ `(*(list + i * sizeof(player))).score`

Page 10: More Questions

16. [3 points] Suppose you found a security vulnerability in code that you did not write. What is the first step you should take?

    ◯ Create and use an exploit to prove to the author it is real

    ◯ Publish the vulnerability (in a repository such as CVE)

    ◯ Wait to see if the author finds and fixes it

    ◯ Tell the author of the software

17. [6 points] For each of the following statements about memory, write **H** if it is true about the heap, **S** if it is true about the stack, **B** if it is true about both, or **N** for neither.

    ____ Is used to keep track of function calls.

    ____ Can be used for arrays and structs.

    ____ Space may be requested using `calloc()`.

    ____ Requires `free()` to avoid memory leaks.

    ____ Contains the code of our program.

    ____ Contains string literals defined in our program.

18. [3 points] Consider the following code:

```
int main() {
    printf("Do. ");
    write(1,"Or do not. ", 11);
    puts("There is no try. -Yoda ");
}
```

What will be printed to standard out?

◯ Do. Or do not. There is no try. -Yoda

◯ Or do not. Do. There is no try. -Yoda

◯ Do. There is no try. -Yoda

◯ Do. There is no try. -Yoda Or do not.

Page 11: More Questions

For the next two questions, consider the following C code that connects to a socket and writes the contents of a buffer to the socket.

```
1  void connectWrite() {
2      const char *buf = "Use the force!";
3      struct sockaddr_in ip;
4      memset(&ip, 0, __________);
5      ip.sin_family = AF_INET;
6      ip.sin_port = htons(55555);
7      ip.sin_addr.s_addr = inet_addr("127.0.0.1");
8      int s = socket(AF_INET, SOCK_STREAM, 0);
9      int res = connect(s, (struct sockaddr *) &ip, __________);
10     if (!res) {
11         ____________________________
12     }
13     close(s);
14 }
```

19. [3 points] In lines 4 and 9, we must provide the correct size of the `ip` struct to `memset()` and `connect()`, respectively. Which of the following could be used to calculate the size? (Fill in the circle completely for all that apply.)

- ◯ `sizeof(*ip)`
- ◯ `sizeof(&ip)`
- ◯ `sizeof(ip)`
- ◯ `sizeof(sockaddr_in)`
- ◯ `sizeof(struct sockaddr_in)`

20. [3 points] Which of the following could we use in the blank on line 11 to correctly write the buffer `buf` over the socket?

- ◯ `puts("%s", buf);`
- ◯ `write(2, buf, strlen(buf));`
- ◯ `write(s, buf, strlen(buf));`
- ◯ `write(s, &buf, strlen(buf));`
- ◯ `write(0, &buf, strlen(buf));`

Page 12: Free Points, Extra Credit, Assumptions

21. [2 points] **(Free Points)** We covered a lot of material in the course this semester, from electricity on wires through most of C. What is one topic you wished we had spent more time discussing in class or lab?

22. [2 points] **(Extra Credit)** My father was 86 years old when he passed last semester, and he was never great with computers. Explain your favorite part of the course material to him in a way that he could appreciate and understand. (Only serious answers will receive extra credit.)

Assumptions. Assume unless otherwise specified:

- All necessary `#include`s have been used
- `char`, `short`, `int`, and `long` are 8-, 16-, 32-, and 64-bits long, respectively
- The compiler pads pointers where it is allowed to do so such that `sizeof(struct X)` is:
  - an even multiple of the size of its largest field
  - the smallest such multiple big enough to store all its fields

Notes: Callee-saved registers: `%rbx`, `%rsp`, `%rbp`, `%r12`, `%r13`, `%r14`, `%r15`

Page 13: Man Page

STRSEP(3) Linux Programmer's Manual STRSEP(3)

NAME

`strsep` - extract token from string

SYNOPSIS

```
#include <string.h>

char *strsep(char **stringp, const char *delim);
```

DESCRIPTION

If `*stringp` is `NULL`, the `strsep()` function returns `NULL` and does nothing else. Otherwise, this function finds the first token in the string `*stringp`, that is delimited by one of the bytes in the string delim. This token is terminated by overwriting the delimiter with a null byte (`'\0'`), and `*stringp` is updated to point past the token. In case no delimiter was found, the token is taken to be the entire string `*stringp`, and `*stringp` is made `NULL`.

RETURN VALUE

The `strsep()` function returns a pointer to the token, that is, it returns the original value of `*stringp`.

Page 14: Man Page

CSOSTRCMP(3) Linux Programmer's Manual CSOSTRCMP(3)

NAME

`csostrcmp, csostrncmp` - compare two strings

SYNOPSIS

```
#include <cso1final.h>

int csostrcmp(const char *s1, const char *s2);

int csostrncmp(const char *s1, const char *s2, size_t n);
```

DESCRIPTION

The `csostrcmp()` function compares the two strings `s1` and `s2`. The comparison is done using unsigned characters.

`csostrcmp()` returns an integer indicating the result of the comparison, as follows:

- `0`, if the `s1` and `s2` are equal;
- `-1`, if `s1` is less than `s2`;
- `1`, if `s1` is greater than `s2`.

The `csostrncmp()` function is similar, except it compares only the first (at most) `n` bytes of `s1` and `s2`.

RETURN VALUE

The `csostrcmp()` and `csostrncmp()` functions return integers `-1`, `0`, or `1` if `s1` (or the first `n` bytes thereof) is found, respectively, to be less than, to match, or be greater than `s2`.

Page 15: Our ISA

Our Example ISA. This is the ISA described in class and used in Lab 4-5 and Homework 3-4. Each instruction is one (or two) bytes, defined as:

If the reserved bit (bit 7) in our instruction is 0, the following table defines our instruction encoding.

icode	b	behavior
0		`rA = rB`
1		`rA += rB`
2		`rA &= rB`
3		`rA` = read from memory at address `rB`
4		write `rA` to memory at address `rB`
5	0	`rA = ~rA`
	1	`rA = -rA`
	2	`rA = !rA`
	3	`rA = pc`
6	0	`rA` = read from memory at `pc + 1`
	1	`rA` += read from memory at `pc + 1`
	2	`rA` &= read from memory at `pc + 1`
	3	`rA` = read from memory at the address stored at `pc + 1`
		For icode 6, increase `pc` by 2 at end of instruction
7		Compare `rA` as 8-bit two's-complement to `0`
		if `rA <= 0` set `pc = rB`
		else increment `pc` as normal

If the reserved bit (bit 7) in our instruction is 1, the following table defines our extended instruction encoding to provide function calls and stack operations.

icode	b	behavior
0	0	Decrement `rsp` and push contents of `rA` to the stack
	1	Pop the top value from the stack into `rA` and increment `rsp`
	2	Push `pc + 2` onto the stack, `pc` = read from memory at `pc + 1`
	3	`pc` = pop the top value from the stack
		For icode 0, if `b` is not 2, update the `pc` as normal

Calling conventions: Register 1 is callee-save. Function arguments will be placed in registers 2 and 3; the return value is stored in register 0.

Page 16: Scratch Paper

Nothing on this page will be graded.