COA1 Final Exam – Fall 2018

Name: _____ Computing ID: _____

Write clearly: if we are unsure of what you wrote, or you wrote so faintly we cannot read it after scanning your test, you will get a zero on that question.
Pledge the exam or you will lose points.
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
- compilation happens using `clang` on an x86-64 Linux system

Single-select by default: Multiple select are all clearly marked; answer them by putting 1 or more letters in the box, or writing "`none`" if none should be selected.
Mark clarifications: If you need to clarify an answer, do so, and also add a `*` to the top right corner of your answer box or blank.
Reference pages: There are reference pages at the end of this exam. You are welcome to remove these, write on them, etc. Do turn them in, along with the exam and any scratch paper you use.

.....

Question 1 [2 pt]: What is `0xF08FC023 | 0x12345678`? Answer in hexadecimal.

Question 2 [2 pt]: For which of the following is it possible for `x == y` to be true even if `x` and `y` are stored as different bit sequences?

Select all that apply by putting one or more letters in the box.

A 32-bit 2's complement integers
B 32-bit biassed integers
C 32-bit IEEE-style floating-point numbers
D 32-bit sign bit integers
E 32-bit unsigned integers

Answer:

Question 3 [2 pt]: Write a C expression equivalent to `x & y` without using `&`.

Question 4 [2 pt]: Provide a value (in hex or binary) of `x` for which `(x << 2) >> 1` does not equal `x << 1`. Assume `x` is a `signed char`.

Question 5 [2 pt]: Draw a 2-input adder circuit: that is, a set of logic gates with 2 input wires (x and y) and two output wires (c and s) such that $2c + s$ (a value between 00_2 and 10_2) is the sum $x + y$.

Question 6 [2 pt]: Suppose `int *x` has value `0x108` and we execute the instruction `*x = 0x12345678`. Fill in the hex bytes set by this operation, leaving unmodified bytes blank.

Address:	103	104	105	106	107	108	109	10A	10B	10C	10D
Value:											

Question 7 [2 pt]: Suppose we wanted to extend our toy ISA (documented at the end of this exam) to use 4-byte program registers and addresses instead of 1-byte program registers and addresses. Which of the following will no longer be true after this change?

Select all that apply by putting one or more letters in the box.

A All instructions (except 6) are encoded in one byte
B Instruction 6 adds 2 to `pc`
C Instructions 3 and 4 use a program register as an address
D Instructions 5.3 and 7 assume program registers and the `pc` are the same size
E Most instructions add 1 to `pc`
F We have 256 bytes of memory

Answer:

Question 8 [2 pt]: Fill in the blank so that the code prints `true`.

```
#define N _____
if(sizeof(short[N]) == sizeof(short *))
    puts("true");
else
    puts("false");
```

Question 9 [4 pt]: Re-write this code using labels and `goto`s instead of a loop.

```
for(int i=0; i<n; i+=1)
    printf("Step %d\n", i+1);
```

Question 10 [2 pt]: Insert a `free` invocation into one of the blank lines in the following code to remove the memory leak.

```
typedef struct ll_s { struct ll_s *next; int value; } ll;
ll *push(ll *old, int value) {


    ll *ans = malloc(sizeof(ll));
    ans->next = old;
    ans->value = value;


    return ans;
}
ll *pop(ll *old, int *value) {


    ll *ans = old->next;
    if (value) *value = old->value;


    return ans;
}
```

Question 11 [6 pt]: Convert the following function into x86-64 assembly. Use the standard C on Linux calling conventions.

```
long f(long a, long b) {
    if (b == 0) return a;
    return f(b, a-b);
}
```

```
f:
```

Information for questions 12–13
The following code exercises undefined behavior, even if all arguments are correctly provided

```
double *vecsum(const double **list_of_vectors, size_t num, size_t len) {
    double *sum = malloc(sizeof(double)*len);
    for(int veci = 0; veci < num; veci += 1) {
        for(int i = 0; i < len; i += 1) {
            sum[i] += list_of_vectors[veci][i];
        }
    }
    return sum;
}
```

Question 12 [2 pt]: (see above) What is the undefined behavior and what problem does it cause if it does not behave the way the function writer intended?

Question 13 [4 pt]: (see above) What does this function do if the undefined behavior does not cause a problem? Include both an example and a description, such as might go in the **description** and **examples** sections of a manual page.

Question 14 [8 pt]: Write an implementation of `strtol` using no library functions except `isspace`. Your implementation must work for any `base` between 2 and 10; you do not need to handle `0` bases or `0x` prefixes, but must meet the rest of the specification given in the manual page excerpt at the end of this exam.

If you get stuck on one part of the solution, enter a comment or a partial solution and complete the rest.

```
long int strtol(const char *nptr, char **endptr, int base) {
```

Information for questions 15–17
Pick one library function for reading from a file. The next several questions are about that function.

What is the name of the function you picked? _____

Question 15 [2 pt]: (see above) This function accepts the file as a

A file descriptor (`int`)
B path name (`char *`)
C stream (`FILE *`)

Answer:

Question 16 [2 pt]: (see above) Each invocation of this function provides the bytes it reads

A directly in the return value
B in a `char *` that it `malloc`ed
C in a `char *` that it was passed as an argument
D other: _____

Answer:

Question 17 [1 pt]: (see above) The display function `puts` requires a null-terminated string. If you fill an array of characters using this function (possibly with multiple invocations if it cannot fill an array in one go), do you need to manually add the `'\0'` before `puts` can display it?

A no, the function provides its own `'\0'`
B yes, you must add the `'\0'` manually

Answer:

Information for questions 18–19
Garbage is defined as _____ memory that _____

Question 18 [2 pt]: (see above) Which of the following (separated by "or"s) need to be in the first blank above?

Select all that apply by putting one or more letters in the box.

A global
B heap
C `malloc`ed
D stack

Answer:

Question 19 [2 pt]: (see above) Which of the following (separated by "and"s) need to be in the second blank above?

Select all that apply by putting one or more letters in the box.

A has meaningful or uncleared data
B has meaningless or uninitialized data
C has not been `free`d
D the program has no pointers to
E will not be used by the program in the future

Answer:

Information for questions 20–21
For each of the following, identify the bug in the code and how to fix it. Assume that all variables the code uses without defining were defined and correctly initialized earlier in the code.

Question 20 [2 pt]: (see above)

```
size_t need = have + 1;
realloc(x, sizeof(int)*need);
x[have] = new;
```

Question 21 [2 pt]: (see above)

```
struct p { double x; double y; };
struct p *array = malloc(n * sizeof(struct p *));
for (int i=0; i<n; i+=1) array[i].x = 2.3
```

Pledge:

On my honor as a student, I have neither given nor received aid on this exam.

Your signature here

Toy ISA

Our toy ISA was defined via the following table. All registers (`r0` through `r3` and `pc`) stored 1-byte values and unless otherwise specified below each instruction added 1 to `pc`.

`icode`	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	do different things for different values of `b`:

`b`	action
0	`rA = ~rA`
1	`rA = -rA`
2	`rA = !rA`
3	`rA = pc`

6	do different things for different values of `b`:

`b`	action
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`

In all 4 cases, increase `pc` by 2, not 1, at the end of this instruction

7	Compare `rA` (as an 8-bit 2's-complement number) to `0`; if `rA <= 0`, set `pc = rB` otherwise, increment `pc` like normal.

Each instruction was encoded in a single byte, with the high-order bit being `0`, followed by the three-bit `icode`, the two-bit value of `A`, and finally the two-bit value of `B` in the low-order bits.

NAME — malloc, free, calloc, realloc

SYNOPSIS

```
#include <stdlib.h>

void *malloc(size_t size);
void free(void *ptr);
void *calloc(size_t nmemb, size_t size);
void *realloc(void *ptr, size_t size);
```

DESCRIPTION

The **malloc**() function allocates *size* bytes and returns a pointer to the allocated memory. *The memory is not initialized.* If *size* is 0, then **malloc**() returns either NULL, or a unique pointer value that can later be successfully passed to **free**().

The **free**() function frees the memory space pointed to by *ptr*, which must have been returned by a previous call to **malloc**(), **calloc**(), or **realloc**(). Otherwise, or if *free(ptr)* has already been called before, undefined behavior occurs. If *ptr* is NULL, no operation is performed.

The **calloc**() function allocates memory for an array of *nmemb* elements of *size* bytes each and returns a pointer to the allocated memory. The memory is set to zero. If *nmemb* or *size* is 0, then **calloc**() returns either NULL, or a unique pointer value that can later be successfully passed to **free**().

The **realloc**() function changes the size of the memory block pointed to by *ptr* to *size* bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes. If the new size is larger than the old size, the added memory will *not* be initialized. If *ptr* is NULL, then the call is equivalent to *malloc(size)*, for all values of *size*; if *size* is equal to zero, and *ptr* is not NULL, then the call is equivalent to *free(ptr)*. Unless *ptr* is NULL, it must have been returned by an earlier call to **malloc**(), **calloc**(), or **realloc**(). If the area pointed to was moved, a *free(ptr)* is done.

RETURN VALUE

The **malloc**() and **calloc**() functions return a pointer to the allocated memory, which is suitably aligned for any built-in type. On error, these functions return NULL. NULL may also be returned by a successful call to **malloc**() with a *size* of zero, or by a successful call to **calloc**() with *nmemb* or *size* equal to zero.

The **free**() function returns no value.

The **realloc**() function returns a pointer to the newly allocated memory, which is suitably aligned for any built-in type and may be different from *ptr*, or NULL if the request fails. If *size* was equal to 0, either NULL or a pointer suitable to be passed to **free**() is returned. If **realloc**() fails, the original block is left untouched; it is not freed or moved.

NAME — strtol

SYNOPSIS

```
#include <stdlib.h>
long int strtol(const char *nptr, char **endptr, int base);
```

DESCRIPTION

The **strtol**() function converts the initial part of the string in *nptr* to a long integer value according to the given *base*, which must be between 2 and 36 inclusive, or be the special value 0.

The string may begin with an arbitrary amount of white space (as determined by **isspace**(3)) followed by a single optional '+' or '-' sign. If *base* is zero or 16, the string may then include a "0x" or "0X" prefix, and the number will be read in base 16; otherwise, a zero *base* is taken as 10 (decimal) unless the next character is '0', in which case it is taken as 8 (octal).

The remainder of the string is converted to a *long int* value in the obvious manner, stopping at the first character which is not a valid digit in the given base. (In bases above 10, the letter 'A' in either uppercase or lowercase represents 10, 'B' represents 11, and so forth, with 'Z' representing 35.)

If *endptr* is not NULL, **strtol**() stores the address of the first invalid character in **endptr*. If there were no digits at all, **strtol**() stores the original value of *nptr* in **endptr* (and returns 0). In particular, if **nptr* is not '\0' but ***endptr* is '\0' on return, the entire string is valid.

RETURN VALUE

The **strtol**() function returns the result of the conversion, unless the value would underflow or overflow. If an underflow occurs, **strtol**() returns **LONG_MIN**. If an overflow occurs, **strtol**() returns **LONG_MAX**. In both cases, *errno* is set to **ERANGE**.

ERRORS

ERANGE The resulting value was out of range.

The implementation may also set *errno* to **EINVAL** in case no conversion was performed (no digits seen, and 0 returned).

NAME – isspace

SYNOPSIS

```
#include <ctype.h>

int isspace(int c);
```

DESCRIPTION

These functions check whether *c*, which must have the value of an *unsigned char* or **EOF**, falls into a certain character class according to the current locale.

isspace() checks for white-space characters. In the `"C"` and `"POSIX"` locales, these are: space, form-feed (`'\f'`), newline (`'\n'`), carriage return (`'\r'`), horizontal tab (`'\t'`), and vertical tab (`'\v'`).

RETURN VALUE

The values returned are nonzero if the character *c* falls into the tested class, and zero if not.