Midterm Exam Answers

CSCI-UA.0201

Instructor: Randy Shepherd
Spring 2017

Section 1: Multiple choice *(select any that apply) - 20* points

01. Representing 10 using the 4 byte unsigned integer encoding and using 4 byte two's complements encoding yields the same bit pattern.

[✓ (a) True]
(b) False

02. Using rounding to the nearest even and rounding to nearest 1/4 (2 bits right of binary point), 10.11100 becomes..

(a) 10.10
[✓ (b) 11.00]
(c) 10.01
(d) 10.00
(e) None of the above

03. Assuming the following is in the context of a well-formed program, what is the result of attempting to compile and execute this code?

```
int* p = NULL;
printf("%d", *p);
```

(a) Will not compile
(b) [✓ Segmentation fault]
(c) 0 will be printed
(d) Garbage will be printed
(e) None of the above

04. In the absence of explicit initialization the following is true about variables in C.

(a) global variables will contain "garbage"
(b) static variables will contain "garbage"
(c) [✓ local variables will contain "garbage"]
(d) All of the above

05. Assuming the following is in the context of a well-formed program which of the logical statements are true?

```
float a = -0.0;
int b = *((int*)&a);
int c = INT_MAX + 1;
```

(a) b != c
(b) b > c
(c) b < c
[✓ (d) b == c]
(e) None of the above

06. Suppose '*int x = 0x00FF;*' and '*int y = 0xFF00*', which of the following evaluates to false if used in an if-statement?

[✓ (a) (x & y)]
(b) (x | y)
(c) Both
(d) Neither

07. Things that must be allocated in heap memory include...

[✓ (a) variables declared in a function that must survive after the function call completes]
(b) variables declared in a function that must retain a value between invocations
(c) variables that are to be globally accessible
(d) None of the above

08. In C, if x is an integer variable, the expression "`x << 3`" computes x * 8 but does not change the value of x.

[✓ (a) True]
(b) False

09. Which of the following types has a largest size (in terms of bytes), assuming 64-bit machine?

(a) int*
(b) float*
(c) double
(d) long
[✓ (e) All are the same size]

10. The following bit vector: 011101 can be interpreted as...

(a) An unsigned binary number
(b) A two's complement binary number
(c) A 6-bit IEEE 754-like encoded floating point number
[✓ (d) All of the above]

Grading Guide: Every question is worth 2 points. Each one only has one correct answer. If they select the correct answer, but additional incorrect answers, reduce by one point for each incorrect answer. In other words, if the student picks two answers, one of them being the correct one, then they get one point. If they select the correct answer and two additional incorrect answers, then they get zero points.

Section 2: Bit Twiddling - 15 points

01. Initialize '`mask`' such that when it is AND'ed with 'a' the results is the last three bits of 'a' (e.g. the 1's 2's and 4's bit).

```
short a = 0b101010101;

short mask = 0b000000111;

assert((a & mask) == 0b000000101);
```

02. Assume **'char x = 0x3B'** and **'char y = 0x77'**

What is the value of z in decimal after the following line of code?

```
char z = (x ^ y) | y;
```

127

03. Assume **'char x = -1'**

What is the value of z in decimal after the following line of code?

```
char z = x >> 7;
```

-1

Grading Guide: These are all or nothing. 5 points per question.

Section 3: C Code Reading - 15 points

01.

```
void foo(int x, int y) {
    int t = x;
    x = y;
    y = t;
}

int main() {
    int x = 1;
    int y = 2;
    printf("x=%i, y=%i\n", x, y);

    foo(x, y);
    printf("x=%i, y=%i\n", x, y);

    return 0;
}
```

The following will be printed...

x=1, y=2

x=1, y=2

02.

```
int main() {
    short a[10] = { 1,2,3,4,5,6,7,8,9,10 };
    unsigned int i;
    for (i = 9; i >= 0; i--)
    {
        printf("index %u ", i);
    }
    printf("\n");
}
```

This will be an infinite loop because the counter i is unsigned. The loop condition is *if i >= 0 then do body and subtract 1 from i* Subtracting 1 from an unsigned int with a value of 0 will overflow to UINT_MAX.

03. (*Hint: What is **guaranteed** to print?*)

```
int main() {
    int x = returns_int();
    float f = returns_float();
    double d = returns_double();

    if(x == (int)(float) x) { printf("A"); }

    if(x == (int)(double) x) { printf("B"); }

    if(f == (float)(double) f) { printf("C"); }

    if(d == (double)(float) d) { printf("D"); }

    if(x + x >= 0) { printf("E"); }

    if(d * d >= 0.0) { printf("F"); }

    if((d+f)-d == f) { printf("G"); }

    return 0;
}
```

The following will always be printed...

BCF

Grading Guide: The first two are all or nothing.

The third one, if they include in their answer A, D or G they must explain:
A: The value of x must be small enough to fit into the 23 frac bits of a float.
D: The value of f must be small enough to fit into the 54 frac bits of a double.
G: The values of d and f must be relatively "close" in magnitude. In other words, d and f must not be wildly different in terms of the size of the numbers they represent.

If they answer in this way for all three give them full credit. Otherwise take a point off for each incorrect answer.

Section 4: IEEE 754 - 20 points

Consider the following 6-bit floating point representation based on the IEEE floating point format. There is a sign bit ***s*** in the most significant bit. The next three bits are the ***exp***, with an exponent bias of 3. The last two bits are the ***frac***.

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/mantissa and E is the exponent.

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

- $\mathbf{M_2}$: the value of M using binary format
- $\mathbf{M_{10}}$: the value of M using decimal format
- $\mathbf{E_{10}}$: the value of E in decimal format
- $\mathbf{V_{10}}$: the numeric value represented by the binary sequence using decimal format.

Fractions or rationals are fine for base $\mathbf{V_{10}}$.

Use the provided scrap paper for your work at the end of this packet, but enter all the answers into the table.

Bit Vector	$\mathbf{M_2}$	$\mathbf{M_{10}}$	$\mathbf{E_{10}}$	$\mathbf{V_{10}}$
011011	**1.11**	**1.75**	**3**	**14**
100100	**1.00**	**1**	**-2**	**-0.25 -or- -1/4**
100000	**0.00**	**0**	**-2**	**-0**
111111	**N/A -or- NaN**	**N/A -or- NaN**	**N/A -or- NaN**	**NaN**
000011	**0.11**	**0.75**	**-2**	**.1875 -or 3/16**

Grading Guide: One point for each correctly filled cell.

Section 5: Number Systems - 10 points

Fill in the empty boxes in the following table (use the space below for your work, but enter all final answers into the table).

(Hint: You can only use 8-bits!)

Two's Complement 8-bit Binary Number	Decimal
00001010	**10**
10101010	**-86**
11011111	-33

Unsigned 8-bit Binary Number	Hexadecimal
11101010	**OxEA**
Not Possible -or- N/A	0x1B0

^^^ If they wrote the 10110000 binary number there must be some sort of note that they understood that the MSB bit was truncated. In this case give half credit.

Grading Guide: Two points for each correctly filled cell.

Section 6: C Programming - 20 points

In lab1, we searched for, inserted and removed nodes on a linked list. Here we will add a new function to our linked list library.

In this question we will write a function with the following prototype:

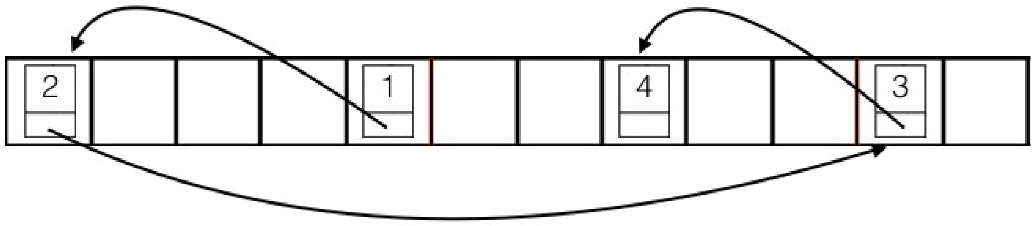
```
struct node* list_reduce(struct node* head);
```

The purpose of this function will be to "reduce" the nodes of the linked list into one *new* list with one *new* node.

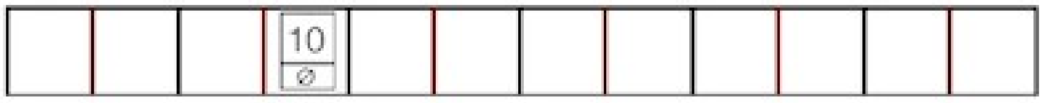
In your program you can use the following linked list node struct, assume it is declared and defined.

```
struct node {
  int value;
  struct node* next;
}
```

So for example, imagine you had a linked list that looked like this in memory, where the head is the node with the value 1:



The list returned would look like this (The return value is a pointer to the node with the value 10):



Notes:

- You not have to write a whole program, no main function required.
- Do not mutate any existing nodes in the list passed as a parameter.
- Manage your memory properly.
- Put your answer on the following, lined page.

```c
struct node* list_reduce(struct node* head) {
    int sum = 0;
    struct node* n = head;
    while(n != NULL) {
        sum += n->value;
        struct node* freeMe = n;
        n = n->next;
        free(freeMe);
    }

    struct node* result = malloc(sizeof(struct node));
    result->value = sum;
    result->next = NULL;

    return result;
}
```

Grading Guide:

Additive points – For each of the below, award the points.

- **3 pt – using malloc properly to create a new node for the new list**
- **2 pt – looping over the old list properly**
- **1 pt – accessing and populating new struct properly**
- **1 pt – summing the old list properly**
- **2 pt – freeing the old list properly**
- **10 pt – general flow of program. (At a 10k ft level, is the code attempting to do the right things. Does their answer indicate an understanding of the problem and what is required for a solution? Moreover, 5 pts for writing any reasonable C code, then 5 more pts if the code demonstrates conceptual understanding of the problem and the steps required to get there.)**

Note that they don't have to have the exact same answer as me, just something approximately close.