CMSC 15200 section 3 exam1

Nhat Lam

TOTAL POINTS

97 / 100

QUESTION 1

1 p.2: number table 12 / 12

✓ + 1 pts row 2, octal: 17

✓ + 1 pts row 2, decimal: 15

✓ + 1 pts row 2, hex: F

✓ + 1 pts row 3, binary: 111111

✓ + 1 pts row 3, decimal: 63

✓ + 1 pts row 3, hex: 3F

✓ + 1 pts row 4, binary: 11011

✓ + 1 pts row 4, octal: 33

✓ + 1 pts row 4, hex: 1B

✓ + 1 pts row 5, binary: 1100 1010

✓ + 1 pts row 5, octal: 312

✓ + 1 pts row 5, decimal: 202

QUESTION 2

2 p.2: hex to binary 2 / 2

✓ + 2 pts correct: 1010 1011 1100 1110 1111 0011 0010 0001

+ 1 pts mostly correct

QUESTION 3

3 p.2: binary to hex 2 / 2

✓ + 2 pts correct: 83C4 FD57

+ 1 pts mostly correct

QUESTION 4

4 p.2: signed int 3 / 3

✓ + 3 pts correct: -2^31 + 2^27 + 2^23 + 2^2

+ 2 pts correct except for the sign of 2^31

+ 1.5 pts off by one error

+ 0 pts incorrect

QUESTION 5

5 p.3: bit operations 6 / 6

✓ + 1 pts c1 is -4

✓ + 1 pts c2 is -1, but *everyone gets this point because this is an unintentional trick question*! (In context, your brain wants this to be b2 >> 2, not b1 >> 2, and may perceive it as such.)

✓ + 1 pts c3 is 8

✓ + 1 pts c4 is 255

✓ + 1 pts c5 is 247

✓ + 1 pts c6 is 192

QUESTION 6

6 p.3: ++ and -- 3 / 4

✓ + 1 pts j1 is 3

✓ + 1 pts j2 is 7

✓ + 1 pts j3 is 7

+ 1 pts j4 is 51

QUESTION 7

7 p.3: pointer arithmetic 3 / 3

✓ + 1 pts pc+1 is 1025

✓ + 1 pts pi+10 is 2088

✓ + 1 pts pl+10 is 4336

+ 0 pts Graded

QUESTION 8

8 p.4: ctr1 5 / 5

✓ - 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.

- 0.5 pts Half the width should be added, not subtracted, to computer the center's x coordinate.

- 0.5 pts Half the height should be subtracted, not added, to computer the center's y coordinate.

- 1 pts The box argument is treated as a pointer to a struct.

- 1 pts A pointer is returned.

- 1 pts A pointer to a stack object is returned (this is

especially bad).

- 1 pts The center x and center y must be displaced from the upper left x and the upper left y. They are not just half the width and half the height.

- 5 pts no response

QUESTION 9

9 p.4: ctr2 5 / 5

✓ **- 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.**

- 0.5 pts Half the width should be added, not subtracted, to computer the center's x coordinate.

- 0.5 pts Half the height should be subtracted, not added, to computer the center's y coordinate.

- 1 pts The box argument is not treated as a pointer to a struct.

- 1 pts A pointer is not returned.

- 2 pts A pointer to a stack object is returned (this is especially bad). The stack data will not persist beyond the function call. You should allocate a new point struct on the heap and return a pointer to that.

- 1 pts The center x and center y must be displaced from the upper left x and the upper left y. They are not just half the width and half the height.

- 1 pts Treated upper_left as a pointer.

- 2 pts No heap space is allocated for the center. You need to allocate with malloc(sizeof(struct point)).

- 5 pts no response

QUESTION 10

10 p.5: scale (malloc) 5 / 5

✓ **- 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.**

- 2 pts No heap space is allocated for the result. You need to allocate with malloc(sizeof(struct vec3)).

- 2 pts A pointer to a stack object is returned. The stack data will not persist beyond the function call. You should allocate a new point struct on the heap and return a pointer to that.

- 0 pts No new vector is created, as specified by the return type, and type of v. This is not what we intended, but it is also not incorrect.

- 2 pts You return a struct directly and not a pointer to struct.

- 5 pts no response

QUESTION 11

11 p.5: scale (void) 5 / 5

✓ **- 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.**

- 2.5 pts The modifications made are all local and do not persist beyond the lifetime of the function.

- 5 pts no response

QUESTION 12

12 p.5: dot product 6 / 6

✓ **- 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.**

- 6 pts no response

QUESTION 13

13 p.6: Fibonacci without recursion 10 / 10

✓ **- 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.**

- 5 pts Miscomputes the fibonacci sequence using fib(n) = (i-2) + (i-1) with "for loop" indices instead of a fib(n) = fib(n-2) + fib(n-1).

- 2 pts Solution does not keep track of previous fib results.

- 1 pts Result variable returned is not defined as an unsigned long.

- 8 pts Solution is correct, but instructions forbid use of recursion.

- 10 pts incorrect response

- 10 pts no response

QUESTION 14

14 p.7: binary string 8 / 10

- 0 pts The response is entirely correct or

essentially correct subject to deductions enumerated below.

- **2 pts** Fails to set null terminator.
- **1 pts** Null terminator must be written into position 8 of the string.
- **3 pts** Bits in reverse order.
- **1 pts** Unnecessary call to atoi().
- **3 pts** Didn't allocate any space for string.
- **2 pts** Checks if a shifted bit == 1, rather than checking for != 0.
- **10 pts** no response

- 2 Point adjustment

If n is small (ie. the most significant bit is not 1), n will reach 0 before you set a value at every index of your array. These unset indexes will just be garbage.

QUESTION 15

15 p.8: to_poly 12 / 12

✓ **- 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.**

- **4 pts** Uses an array of points, instead of an array of pointers to points.
- **2 pts** No space is allocated for the polygon struct.
- **2 pts** No space is allocated for the array of pointers.
- **3 pts** No space is allocated for the points pointed to.
- **2 pts** The points must be heap-allocated (see the instructions).
- **1 pts** Allocating space with a constant instead of sizeof, or performing arithmetic to compute a struct size manually.
- **1 pts** Allocating with sizeof(struct point) instead of sizeof(struct point *). This is not likely to cause a problem at runtime, because it is enough space, but it is nonetheless incorrect because it is wasteful of space as well as confusing.
- **1 pts** Using dot rather than arrow to access pointer-to-struct data.
- **1 pts** "n" field of polygon struct not set.
- **1 pts** Indexes the point array starting at 1, not 0.
- **12 pts** no response
- **0.5 pts** Incorrect arithmetic

QUESTION 16

16 p.9: polygon_free 10 / 10

✓ **- 0 pts The response is entirely correct or essentially correct subject to deductions enumerated below.**

- **4 pts** Using a pointer that has been freed. Make it a practice never to use pointed-to data that has already been freed.
- **3 pts** Not freeing the individual points in "points".
- **3 pts** Not freeing the array "points" in the polygon.
- **4 pts** Not freeing the polygon struct.
- **3 pts** Not freeing the pointer to the points array.
- **2 pts** Not correctly accessing "points" (poly->points).
- **2 pts** Freeing numbers like poly->n and/or x and y coordinates. These don't need and shouldn't have their own free actions.
- **10 pts** no response
- **10 pts** incorrect response

This exam consists of 10 numbered pages, including this page and a blank page at the end of the test if more space is needed for any response.

Please write your name here: Jarvis Lam
(Nhat Lam in Canvas)

Also, please write your initials in the upper left corner of each page. Since these tests will be scanned, please keep your responses far from the edges of each page to ensure that we can see everything you wrote.

So as not to favor any individual student, and to make the exam-taking experience as uniform as possible, you may not ask questions during the exam.

You may write helper functions. You may use any function you write anywhere else on the test. Please help us by writing a page number when the function in question is not on the same page where it's used.

On this exam, when you allocate memory in written code, you are permitted not to check whether or not the allocation has succeeded.

Throughout this exam, assume that a short int is 2 bytes in size, an int is 4 bytes, a long int is 8 bytes, a float is 4 bytes, a double is 8 bytes, and a pointer is 8 bytes.

You do not need to write #include directives for stdio or stdlib. Assume stdio and stdlib functions are simply available whenever you need them. You may also use strlen if needed.

Finally, some advice. Do not work through the exam in order from first page to last. Scan all the pages to get an overall sense of what is asked of you, and then make a plan for how to spend your time.

"Sorry for messy handwriting.
Not used to new pen"

J. L

Fill in the empty cells in this table. The first, completed row is given as an example.

binary	octal	decimal	hexadecimal
1011	13	11	B
1111	17	15	F
111111	77	63	3F
11011	33	27	1B
11001010	312	202	CA

Convert this hexadecimal number to a binary number:

ABCEF321

1010 1011 1100 1110 1111 0011 0010 0001

Convert this binary number to a hexadecimal number:

1000 0011 1100 0100 1111 1101 0101 0111

8 3 C 4 F D 5 7

Express the following signed int as a sum of powers of 2:

1000 1000 1000 0000 0000 0000 0000 0100

$-2^{31} + 2^{27} + 2^{23} + 2^{2}$

J. L


```
char b1 = -1;
char b2 = -126;
unsigned char b3 = 63;
unsigned char b4 = 200;

char c1 = b1 << 2;
char c2 = b1 >> 2;
unsigned char c3 = b3 & b4;
unsigned char c4 = b3 | b4;
unsigned char c5 = b3 ^ b4;
unsigned char c6 = ~b3;
```

What are the values of c1 through c6, expressed as decimal numbers?

c1 = -4

c2 = -1 // signed char if shift to right yields 1's if it's negative

c3 = 8

c4 = 255 | c5 = 247 | c6 = 192

```
int j1 = 4;
int j2 = 6;
int j3 = 8;
int j4 = (--j1)+(j2++)*(j3--);
```

After the last statement is complete, what are the values of j1, j2, j3, and j4?

j1 = 3

j2 = 7

j3 = 7

j4 = 17

```
char *pc = malloc(1);
int  *pi = malloc(sizeof(int));
long *pl = malloc(sizeof(long));
```

If the numeric value of pc is 1024, what is the numeric value of pc+1?

1025

If the numeric value of pi is 2048, what is the numeric value of pi+10?

2088

If the numeric value of pl is 4096, what is the numeric value of pl+30?

4336

J. L


```
struct point {
  double x;
  double y;
};

struct axis_box { // An axis box has sides parallel to the axes.
  struct point upper_left;
  double width;
  double height;
};
```

Note the types of ctr1 and ctr2. They compute the same logical result, but the mechanisms are different, namely, in terms of the use of pointers. Implement each function.

```
// return the point at the center of the given axis_box
//   if the box width or height is not positive, behavior is undefined

struct point ctr1(struct axis_box b) {
    struct point out_center;
    out_center.x = b.upper_left.x + (b.width)/2;
    out_center.y = b.upper_left.y - (b.height)/2;
    return out_center;
};
```

```
// return the point at the center of the given axis_box
//   if the box width or height is not positive, behavior is undefined

struct point *ctr2(struct axis_box *b) {
    struct point* out_center = malloc(sizeof(struct point));
    out_center->x = b->upper_left.x + (b->width)/2;
    out_center->y = b->upper_left.y - (b->height)/2;
    return out_center;
}; // free out_center in main()
/*
int main() {
   free(out_center) //after it's ~~assigned~~ used
}*/
// return 0;
```

J. L

A 3-dimensional vector <x,y,z> is represented as follows.

```
struct vec3 {
  double x;
  double y;
  double z;
};
```

When a vector <x,y,z> is multiplied by some scalar s, the result is <sx,sy,sz>. For example, the scalar 10 times the vector <2,3,4> is <20,30,40>.

Implement the following two different version of scale, guided by the types given.

```
struct vec3 *scale(double s, struct vec3 *v) {
    struct vec3 *scale_out = malloc(sizeof(struct vec3));
    scale_out -> x = (v -> x) * s;
    scale_out -> y = (v -> y) * s;
    scale_out -> z = (v -> z) * s;
    return scale_out;
} // free scale_out somewhere in main()
void scale(double s, struct vec3 *v) {
    ~~struct vec3 * scale_out~~
    v -> x *= s;
    v -> y *= s;
    v -> z *= s;
    return;
} // pointer will not be terminated even after function finishes
```

The dot product of two vectors <x,y,z> and <q,r,s> is xq+yr+zs. For example, the dot product of <2,3,4> and <1000,100,10> is 2340. Look carefully at the types.

```
double dot(struct vec3 v1, struct vec3 v2) {
    double dot_out;
    dot_out = (v1.x * v2.x) + (v1.y * v2.y) + (v1.z * v2.z);
    return dot_out;
};
```

J.L

The recursive definition the Fibonacci sequence is as follows:

```
fib(0) = 0
fib(1) = 1
fib(n) = fib(n-2) + fib(n-1)
```

Implement the following function to compute the nth Fibonacci number without using recursion.

```
unsigned long fib(unsigned int n) {
    if (n == 0) {
        return 0;
    if (n == 1) {
        return 1;
    if (n > 1){
        int f0 = 0; unsigned long f0 = 0;
        int f1 = 0; unsigned long f1 = 1;
        int out = 1; unsigned long out = 1;
        for (int i = 2; i <= n; i++) {
            out = f0 + f1;
            f0 = f1;
            f1 = out;
        };
        return out;
};
```

J. L

The function build_binary should return a string of characters '0' and '1' of logical length 8 (representation length 9). The result must be a new heap-allocated string.

```
char *build_binary(unsigned char n) {
    char *out_str = malloc(sizeof(char)*9);

    int index = 7;
    while (n!=0 && index>-1) {
        if (n % 2 == 0) {
            out_str[index] = '0';
        } else if (n % 2 == 1) {
            out_str[index] = '1';
        };
        n = n/2;
        index -= 1;
    };
    out_str[8] = '\0';
    return out_str;
};

//free out_str later in main()
```

"Sorry for messy handwriting" J. L

A polygon is represented by a number of points and an array of points. It is assumed the points are in clockwise order, and the first point does not appear a second time at the end of the array. The array representation is a pointer to a pointer to a point.

```
struct polygon {
  unsigned int n;
  struct point **points;
};
```

```
struct axis_box {
  struct point upper_left;
  double width;
  double height;
};
```

Write a function to consume an axis_box as defined a few pages earlier (the definition is reproduced here for convenience) and build the corresponding polygon. The result must be entirely heap allocated. The behavior in case of a malformed axis box is undefined.

```
struct polygon *to_poly(struct axis_box *b) {
// We assume you are building a polygon contained in
// the axis box, or maybe make the box into a polygon
// Since this is really confusing, we assume you mean making a
// polygon out of this box, which is essentially a rectangle.
  struct polygon * poly_out = malloc(sizeof(struct polygon));
  struct point ** points_ = malloc(sizeof(struct point*) * 4);
  struct point * up_left = malloc(sizeof(struct point));
  struct point * up_right = malloc(sizeof(struct point));
  struct point * down_right = malloc(sizeof(struct point));
  struct point * down_left = malloc(sizeof(struct point));
  upleft->x = b->upper_left.x; upleft->y = b->upper_left.y;
  up_right->x = b->upper_left.x + b->width; up_right->y = b->upper_left.y;
  down_right->x = b->upper_left.x + b->width;
  down_right->y = b->upper_left.y - b->height;
  down_left->x = b->upper_left.x; down_left = b->upper_left.y - b->height;
// Continue on bottom of page 9.
```

clarification: "+b->width"

J.L

Assuming the pointed-to polygon and all its data reside on the heap, free all of it.

```
void polygon_free(struct polygon *poly) {
    free ( *( poly -> points ) );
    free (   *((poly -> points) + 1));
    free (   *((poly -> points) + 2));
    free (   *((poly -> points) + 3));
    free (   poly -> points );
    free (   poly );
re  return;
};
```

```
*(points_) = up_left;
*(points_ + 1) = up_right;
*(points_ + 2) = down_right;
*(points_ + 3) = down_left;
poly_out -> points = points_;
poly_out -> n = 4;
return poly_out;
};
```

// from page 8

J. L

(extra space)

1111

1+2

1+2+4+8

2+8+64+128

74+128 = 202

63|2 31|2 15|2 7|2 3|2 1|2 1|0

10|2 5|2 2|2 1|2 0

9 A B C D E F

10 11 12 13 14 15

12|2 6|2 3|2 1|2 1|0

4+7+8

3+6+8

15

$2^6 = 64$

1 7

$_{6}1_{5}1_{4}0_{3}0_{2}1_{1}0_{0}$

3

27|2 13|2 6|2 3|2 1|2 1|0

1+2+4

8+

7.8+7.1

56+7 = 63

7.8

1.8

2+8+64+128

74+128 202

64

192

63|2 31|2 15|2 7|2 3|2 1|2 1|0

b1 = -1

b2 = -126

b3 = 63

b4 = 200

11111111

= 00111111

= 11001000

c1 = 11111100

c2 = 00111111

c3 = 00001000

c4 = 11111111

c5 = 11110111

c6 = 11000000

8+4+1

13

0 1 2 3 4

0 1 1 2 3 4 5

out 1 2 3

fo 1 1 2

f1 1 2 3 128+64

11001010

64

30

18

200|2 100|2 50|2 25|2 12|2 6|2 3|2 1|2 1|0

15|8 7|8 1|0

63|8 56|7|8 7|7|0

63|16 48|3|16 15|3|0

27|8 3|3|8 3|0

27|16 16|1|16 1|1|0

8*30 = 240

4096

4336

n=6

4096

1920

6016

2048+320

2368

1 1 1 1 1 1 1

0 1 2 3 4 5 6 7

1 1 0 0 1 0 0 0

1 1 1 1 1

0 1 2 3 4 5 6

0 1 1 2 3 5 8

0 1 2 3 4 5 6 7

out 1 2 3 5 8 X

f0 0 1 1 2 3 5

f1 1 1 2 3 5 18