CIS 2107
Computer Systems and Low-Level Programming
Fall 2011
Midterm Solutions

October 25, 2011

Name: ______________________________

Page	Points	Score
1	7	
2	10	
3	8	
4	13	
6	17	
7	4	
8	16	
9	15	
10	10	
Total:	100	

Instructions

The exam is closed book, closed notes. You may *not* use a calculator, cell phone, etc.

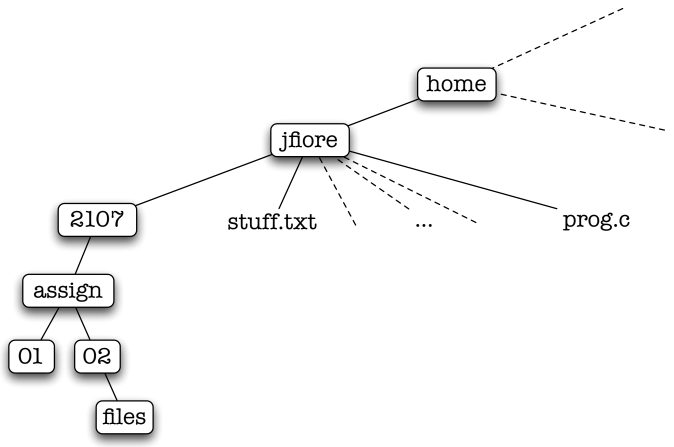
For each of the questions of this quiz, you can assume the following sizes for C data types:

type	**bytes**
char	1
short	2
int	4
long	8
float	4
double	8
void*	4

CIS 2107 Computer Systems and Low-Level Programming — Midterm Solutions, October 25, 2011

For the following questions, you can assume that my home directory is the `jfiore` directory.

1. Unix shell stuff.



(1 point) (a) If I'm in my home directory *i.e.*, `/home/jfiore`, what command can I type in order to run the C compiler on `prog.c`, but not the linker?

Solution: `gcc -c prog.c`

(1 point) (b) If I'm in my home directory, what's the one command that I can type in order to create a `files` directory within the assignment 1 directory?

Solution: `mkdir 2107/assign/01/files`

(1 point) (c) What command can I type to see a list of all of the files in my current directory?

Solution: `ls`

(1 point) (d) If I'm in my home directory, what's the one command that I can type to move `prog.c` to the `files` directory inside assignment 2?

Solution: `mv prog.c 2107/assign/02/files`

(1 point) (e) If I run the command `gcc -E prog.c` to run the preprocessor only on `prog.c`, what does the resulting file contain (*i.e.*, how is it different from `prog.c`)?

Solution: Among other things, the #include statements are replaced with the contents of the files included, and the #define statements are processed.

2. Some conversions.

(1 point) (a) 104 tbytes = ? kbits

(a) **832,000,000,000 or** $104 * 8 * 10$

(1 point) (b) 3 minutes = ? microseconds

(b) $3 * 60 * 10^6$

question 2 continues…

(1 point) (c) 48 mbytes = ? tbytes

(c) **4.8e-05 or** $48/10^6$

(1 point) (d) 128 mbytes = ? kbits

(d) **1,024,000 or 128*8*1000**

(1 point) (e) 1 hour = ? nanoseconds

(e) $60 * 60 * 10^9$

3. Convert 246_{10} to:

(1 point) (a) base 2

Solution:

$$\begin{aligned} 246 &= 123 * 2 + \mathbf{0} \\ 123 &= 61 * 2 + \mathbf{1} \\ 61 &= 30 * 2 + \mathbf{1} \\ 30 &= 15 * 2 + \mathbf{0} \\ 15 &= 7 * 2 + \mathbf{1} \\ 7 &= 3 * 2 + \mathbf{1} \\ 3 &= 1 * 2 + \mathbf{1} \\ 1 &= 0 * 2 + \mathbf{1} \end{aligned}$$

$246_{10} = 11110110_2$

(1 point) (b) base 16

Solution: $246_{10} = 11110110_2 = F6_{16}$

4. Using the approximation trick that we talked about in class, about how much are each of the following?

(1 point) (a) 2^{31}

(a) **2 billion**

(1 point) (b) 2^{29}

(b) **512 million**

(1 point) (c) 2^{43}

(c) **8 trillion**

(2 points) 5. What is $111101011_2 + 11101110_2$ in base 2?

$$\begin{array}{ccccccccccc} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & \\ & & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1_2 \\ + & & & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0_2 \\ \hline & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1_2 \end{array}$$

Which makes sense, because:

$$111101011_2 = 491_{10} 11101110_2 = 238_{10}$$

and $491_{10} + 238_{10} = 729_{10}$, which is 1011011001_2.

(2 points) 6. What is $3967B7_{16} + 2E67_{16}$ in base 16?

$$\begin{array}{rcccccc} & & & ^1 & ^1 & & \\ & 3 & 9 & 6 & 7 & \text{B} & 7_{16} \\ + & & & 2 & \text{E} & 6 & 7_{16} \\ \hline & 3 & 9 & 9 & 6 & 1 & \text{E}_{16} \end{array}$$

7. **data representation**. For these questions, please remember to answer in hex, not binary.

(1 point) (a) In hex, what is the smallest integer that can be represented by a 16-bit two's complement int?

(a) **0x8000**

(1 point) (b) In hex, what is the largest integer that can be represented by a 16-bit two's complement int?

(b) **0x7FFF**

(1 point) (c) In hex, what is the largest integer that can be represented by a 16-bit unsigned int?

(c) **0xFFFF**

(1 point) (d) In hex, what is -1 as a 16-bit two's complement int?

(d) **0xFFFF**

(2 points) 8. What is printed by the following?

```
char x = 50;
char signed_prod;
unsigned char unsigned_prod;

signed_prod = x*3;
unsigned_prod = x*3;

printf("%d\n", signed_prod);
printf("%u\n", unsigned_prod);        /* recall that %u means to print as unsigned */
```

Solution:

```
-106
150
```

Why? When we multiply 50 by 3, we get 150, which is stored as 10010110. When we interpret 10010110 as an unsigned decimal value, it's 150_{10}. When we interpret it as a signed value, it's -106_{10}.

9. **Some bit operations.** If we have `char x = 0x53, y = 0xF9;`, what is the result of the following operations? Your answer must be in the form of exactly two hex digits[1].

solution: The easiest thing to do is to convert to binary first.

0x53=01010011_2 and
0xF9=11111001_2

(1 point) (a) x|y

(a) **0xFB**

(1 point) (b) x||y

(b) **0x01**

(1 point) (c) x<<2

(c) **0x4C**

(1 point) (d) ˜ x

(d) **0xAC**

(1 point) (e) x&0x0F

(e) **0x03**

(1 point) (f) x^y

(f) **0xAA**

(1 point) (g) x&&1

(g) **0x01**

(6 points) 10. For this question, we're doing 5-bit two's complement representation of integers. Fill in the empty boxes in the following table. Addition and subtraction should be performed based on the rules for 5-bit, two's complement arithmetic. Recall that in your book's notation, TMin is defined to be the smallest negative two's complement number that we can represent, and TMax is the largest positive one.

[1]Ignore the possibility of promotion to 32-bit ints. Behave as though we're living in the land of 8-bit arithmetic.

points: ________
out of a possible 13

exam continues…

Name	**Decimal Rep.**	**Binary Rep.**
Zero	0	0 0000
n/a	6	**0 0110**
n/a	−5	**1 1011**
n/a	**-5**	1 1011
n/a	**10**	0 1010
TMax	**15**	**0 1111**
TMin	**-16**	**1 0000**
TMin+2	**-14**	**1 0010**
TMax+2	**-15**	**1 0001**

11. If I have the following:

```
int main(void)
{
  int a=10;
  int b=20;

  int *p=&b;
  int *q=p;

  (*p)++;
  q++;
```

and memory is laid out like this:

q	1000	
p	1004	
b	1008	
a	1012	

Solution: Before the increments, you end up with:

q	1000	1008
p	1004	1008
b	1008	20
a	1012	10

(*p)++ increments the value at address 1008, *i.e.*, 20 becomes 21. q++ increments the pointer q, so we have $1008 + (1)(\texttt{sizeof(int)}) = 1008 + (1)(4) = 1012$. We get:

q	1000	1012
p	1004	1008
b	1008	21
a	1012	10

what do you see if you print:

(1 point) (a) a

(a) **10**

(1 point) (b) &a

(b) **1012**

(1 point) (c) b

(c) **21**

(1 point) (d) &b

(d) **1008**

(1 point) (e) p

(e) **1008**

(1 point) (f) *p

(f) **21**

(1 point) (g) &p

(g) **1004**

(1 point) (h) q

(h) **1012**

(1 point) (i) *q

(i) **10**

(1 point) (j) &q

(j) **1000**

12. fun with floats

(3 points) (a) How would we represent the number 191.6875_{10} in fixed-point binary?

Solution:

- $191_{10} = 10111111_2$
- $0.6875_{10} = 0.1011_2$
- $191.6875_{10} = 10111111.1011_2$

(1 point) (b) Normalize your answer from part (a).

Solution: $10111111.1011_2 = 1.01111111011 * 2^7$

(3 points) (c) How would 191.6875_{10} be stored in a C 32-bit `float` variable? (Remember that for 32-bit floats, the bias value is 127.)

Solution:

sign The number is positive, so the sign bit is 0.

exponent We store the biased exponent. The bias is 127, so we have $7 + 127 = 134_{10} = 10000110_2$. So 10000110 is stored in the exponent field.

mantissa We had $1.01111111011 * 2^7$, so in the mantissa field, we store 01111111011 and then pad the remaining bits of the field with 0s. Remember that we have 32-bit floats. 1 bit was used for the sign, 8 bits were used for the exponent. This leaves $32 - (1 + 8) = 23$ bits for the mantissa. In the mantissa field, we store 01111111011000000000000.

So we end up with:

sign	exponent	mantissa
0	10000110	01111111011000000000000

13. **Recognizing the value of a floating-point variable.** In this question, consider 6-bit floating-point numbers. Two bits are used for the mantissa and 3 bits for the exponent.

(1 point) (a) For a 3-bit exponent field, what is the bias?

Solution: bias $= 2^{3-1} - 1 = 2^2 - 1 = 4 - 1 = 3$

(3 points) (b) What floating-point value does the bit string 0 01 001 represent, where 0 is the sign bit, 01 is the mantissa and 001 is the exponent? Please show all work.

Solution:

- the sign bit is 0, so the number is positive
- the exponent field is 3 bits, so the bias is $2^{3-1} - 1 = 2^2 - 1 = 4 - 1 = 3$
- the exponent field is not all 0s, and it's not all 1s, so it's a normalized number.
- This means that the mantissa has an implied leading `1.`, and that the exponent is the value in the exponent field minus the bias.

bit repr	e	E	2^E	f	M	$2^E * M$	V	Decimal
0 01 001	1	-2	1/4	1/4	5/4	5/16	5/16	0.312500

14. Use the following code to answer the questions.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

typedef struct {
  int x;
  int *p;
  int A[3];
} Stuff;

void func01(int[]);
void func02(char*);
void func03(char*);
void func04(Stuff);

int main(void) {
  int A[]={10,20,30};
  int i=40;
  Stuff s;
  char msg[100];

  s.x=50;
  s.p=&i;
  s.A[0]=60;

  strcpy(msg, "aspirin");

  func01(A);
```

question 14 continues...

```
  func02(msg);
  func03(msg);
  func04(s);

  return 0;
}

void func01(int A[]) {
  A[0]++;
}

void func02(char *s) {
  strcpy(s, "half way there");
}

void func03(char *s) {
  s = malloc(10);
  strcpy(s, "coffee");
}

void func04(Stuff s) {
  s.x=5555;
  *(s.p)=4040;
  s.p=(int*)malloc(sizeof(int));
  *(s.p)=4444;
  s.A[0]=6666;
}
```

(1 point) (a) How many bytes are passed to the function `func01( )`?

(a) **4**

(1 point) (b) How many bytes are passed to the function `func02( )`?

(b) **4**

(1 point) (c) How many bytes are passed to the function `func04( )`?

(c) **20**

What is the value of each of the following after `func04( )` has been called?

(1 point) (d) A[0]

(d) **11**

(1 point) (e) i

(e) **4040**

(1 point) (f) s.x

(f) **50**

(1 point) (g) *(s.p)

(g) **4040**

(1 point) (h) s.A[0]

(h) **60**

(1 point) (i) msg (What's the string?)

(i) `half way there`

(7 points) 15. Write a function which is passed an `int A[ ]` of positive integers and `A`'s length. The function returns the largest item in `A`. Do not use the `[ ]` operator.

Solution:

```
int largest(int A[], int len) {
  int i;
  int l;                          /* largest so far */
  int *c;                          /* current */

  if (len<=0)
    return -1;

  c=A;
  l=*c;
  c++;
  for (i=1; i<len; i++, c++) {
    if (*c>l)
      l=*c;
  }
  return l;
}
```

16. big-endian/little-endian

(7 points) (a) Write a function called `is_big_endian( )` which returns 1 if the machine running the function is big endian or 0 otherwise.

Solution: One possibility:

```
int is_big_endian()
{
  int x = 0xFF;
  char *p=(char*)&x;

  return !((*p)&0xFF);
}
```

(1 point) (b) If we're on a 32-bit little-endian machine and `int x` has the value `0x01234567`, and `x` is stored starting at address 1000, what is the value of the byte stored at address 1002?

Solution: If it's little-endian, we'd have:

address	value
1000	0x67
1001	0x45
1002	0x23
1003	0x01

so, the answer is 0x23

(7 points) 17. Write a function `caps( )` which is passed a string `s`. The function returns a copy of `s`, but with the first letter of each word capitalized. If `s` is NULL or if an error is encountered, the function returns NULL. It is up to the caller to free any memory allocated by `caps( )`.

Solution:

```
#include <string.h>
#include <ctype.h>
#include <stdlib.h>

char *caps(char *s) {
  int len, i;
  char *r;

  if ((len=strlen(s))==0)
    return NULL;

  if ((r=(char*)malloc(len+1))==NULL)
    return NULL;

  r[0]=toupper(s[0]);
  for (i=1; i<len; i++) {
    if (!isspace(s[i]) && isspace(s[i-1]))
      r[i]=toupper(s[i]);
    else
      r[i]=s[i];
  }

  r[i]='\0';
  return r;
}
```

(10 points) 18. Write a program which reads text from STDIN until EOF is entered. The program prints the length of the longest word and the longest line in the file. (Be careful. You don't need to print the longest line and longest word – just their length.)

Solution: One possibility:

```
#include <stdio.h>
#include <ctype.h>

#define OUT 0
#define IN 1

int main(void) {
  int c,                         /* current character */
    l_long_line,                 /* length of longest line */
    l_long_word,                 /* length of longest word */
    l_cur_word,                         /* length of current word */
    l_cur_line,                         /* length of current line */
    state;                          /* state: in word or out of word? */

  l_long_line=l_long_word=l_cur_word=l_cur_line=0;
  state=OUT;
```

points: ________
out of a possible 10

question 18 continues. . .

```

  while ((c=getchar())!=EOF) {
    if (isspace(c)) {
      if (state==IN) {
        if (l_cur_word>l_long_word)
          l_long_word=l_cur_word;
        state=OUT;
        l_cur_word=0;
      }
      if (c=='\n') {
        if (l_cur_line>l_long_line) {
          l_long_line=l_cur_line;
        }
        l_cur_line=-1;
      }
    } else {                        /* we didn't read a space */
      if (state==OUT) {
        state=IN;
        l_cur_word=1;
      } else
        l_cur_word++;
    }
    l_cur_line++;
  }

  printf("length of: longest line = %d, longest word = %d\n",
         l_long_line, l_long_word);
  return 0;
}
```