CS360 Midterm Exam. March 13, 2012. James S. Plank

Put your answers on the answer sheets provided. Do not answer on this exam.

Question 0

Write a procedure **atos()** which takes a NULL-terminated array of strings as its parameter and returns a string. What it should do is allocate, construct and return a single string composed of each string in the array separated by a space. The procedure should run in *O(n)* time, where *n* is the total number of characters in the string that you return.

Question 1

In your **jtar** program, you called **lstat()**, and it filled in a data structure of type **struct stat**. List for me all of the ways in which that data structure was used by your **jtar** program. There may be parts of the data structure that were used for multiple purposes -- list each of these separately.

Question 2

Suppose **rv**, **fd** and **sz** are integers and **buf** is a pointer; and suppose I have the following line in my program:

```
rv = read(fd, buf, sz);
```

Below are 25 potential outcomes of the **read()** call. For each outcome, label it either "P" for "Possible" or "I" for "Impossible." In other words, if it is possible for the outcome to occur, label it "P". If there is no way for the outcome to occur, label it "I". I don't want explanation. I just want P's and I's.

A Fewer than **sz** bytes are read from a file to **buf**, and **rv** is set to the number of bytes that were read.

B **sbrk(0)-buf** is less than **sz**, and as a result, the read generates a segmentation violation

C **fd** is not an open file, and the read call generates a segmentation violation.

D **fd** is a file opened for writing only, and the read call returns -1 as a result.

E **buf** is pointing to a chunk of memory that is fewer than **sz** bytes, and the read call generates a segmentation violation.

F **buf** is pointing to the stack segment and **sz** bytes are read successfully.

G Fewer than **sz** bytes are read from a file to **buf**, and **rv** is set to -1.

H **buf** is pointing into the void and the read call returns -1.

I **buf** is pointing to a chunk of memory that is fewer than **sz** bytes, and the read call corrupts memory in the process.

J **buf** is pointing to a region of **sz** bytes in the globals segment,and the read call returns -1 because of where **buf** is pointing

K A bus error occurs because **buf** is not a multiple of four.

L **buf** is pointing to **sz** bytes in the code segment, and the read call generates a segmentation violation because of where **buf** is pointing.

M Zero bytes are read from any file, and **rv** is set to 0.

N **buf** is pointing to the code segment and **sz** bytes are read successfully.

O **sbrk(0)-buf** is less than **sz**, and as a result, the read call returns -1

P **sz** bytes are read from a file to **buf**, and **rv** is set to **sz**.

Q **fd** is a file opened for writing only, and the read call generates a segmentation violation.

R **buf** is pointing to the stack segment and a segmentation violation occurs because of where **buf** is pointing

S A buffer overflow attack occurs as a result of the read statement.

T **buf** is pointing to a region of more than **sz** bytes in the globals segment and **sz** bytes are read successfully.

U **buf** is pointing to a region of more than **sz** bytes in the globals segment and a segmentation violation occurs because of where **buf** is pointing.

V **fd** is not an open file, and the read call returns -1 as a result.

W **buf** is pointing to the stack segment and the read call returns -1 because of where **buf** is pointing

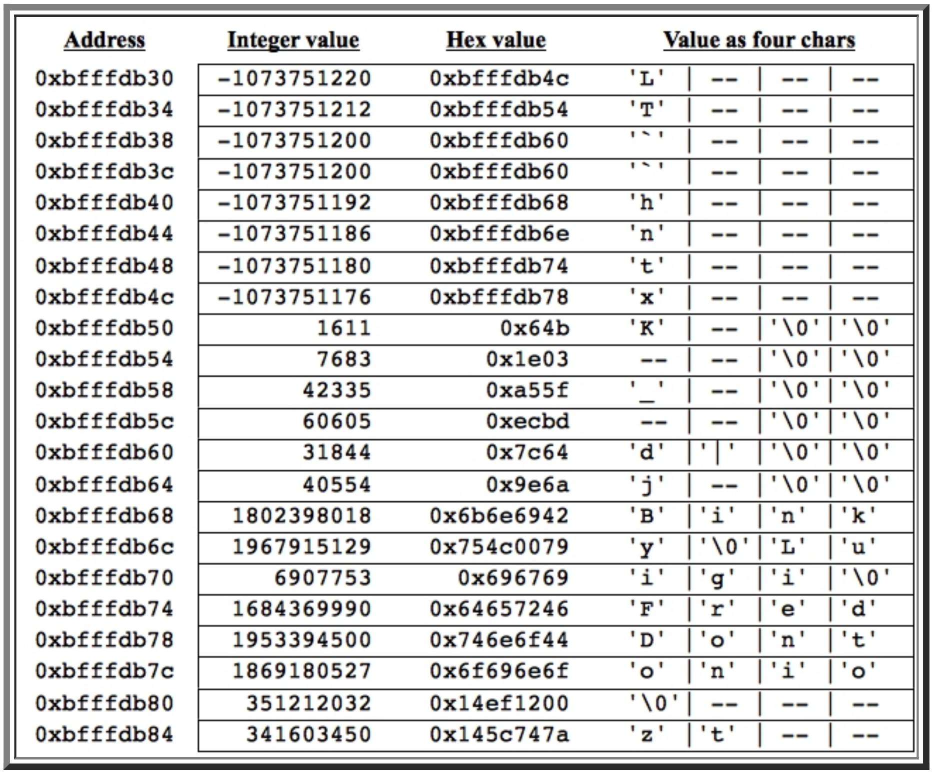
X **buf** is pointing to **sz** bytes in the code segment, and the read call returns -1 because of where **buf** is pointing.

Y **buf** is pointing into the void and the read call generates a segmentation violation.

Question 3

When the procedure **messy_proc()**, is called, the state of memory from addresses 0xbfffdb30 to 0xbfffdb87 is pictured below. In the picture, I show the value of every four bytes in three ways -- I show the value as an integer, in hexadecimal, and as four characters. If the character is not a printable character or the NULL character, I show that with "--".

For example, the four bytes starting at address 0xbfffdb30 are equal to -1073751220 when represented as an integer. They are equal to 0xbfffdb4c when represented as hexadecimal. The byte at 0xbfffdb30 is equal to the 'L' character. The bytes at 0xbfffdb31, 0xbfffdb32 and 0xbfffdb33 are all non-printable characters.



Address	Integer value	Hex value	Value as four chars			
0xbfffdb30	-1073751220	0xbfffdb4c	'L'	--	--	--
0xbfffdb34	-1073751212	0xbfffdb54	'T'	--	--	--
0xbfffdb38	-1073751200	0xbfffdb60	'`'	--	--	--
0xbfffdb3c	-1073751200	0xbfffdb60	'`'	--	--	--
0xbfffdb40	-1073751192	0xbfffdb68	'h'	--	--	--
0xbfffdb44	-1073751186	0xbfffdb6e	'n'	--	--	--
0xbfffdb48	-1073751180	0xbfffdb74	't'	--	--	--
0xbfffdb4c	-1073751176	0xbfffdb78	'x'	--	--	--
0xbfffdb50	1611	0x64b	'K'	--	'\0'	'\0'
0xbfffdb54	7683	0x1e03	--	--	'\0'	'\0'
0xbfffdb58	42335	0xa55f	'_'	--	'\0'	'\0'
0xbfffdb5c	60605	0xecbd	--	--	'\0'	'\0'
0xbfffdb60	31844	0x7c64	'd'	'|'	'\0'	'\0'
0xbfffdb64	40554	0x9e6a	'j'	--	'\0'	'\0'
0xbfffdb68	1802398018	0x6b6e6942	'B'	'i'	'n'	'k'
0xbfffdb6c	1967915129	0x754c0079	'y'	'\0'	'L'	'u'
0xbfffdb70	6907753	0x696769	'i'	'g'	'i'	'\0'
0xbfffdb74	1684369990	0x64657246	'F'	'r'	'e'	'd'
0xbfffdb78	1953394500	0x746e6f44	'D'	'o'	'n'	't'
0xbfffdb7c	1869180527	0x6f696e6f	'o'	'n'	'i'	'o'
0xbfffdb80	351212032	0x14ef1200	'\0'	--	--	--
0xbfffdb84	341603450	0x145c747a	'z'	't'	--	--

Here is **messy_proc()**:

```
void messy_proc(int **a, int *b, char **c)
{
  int i, j;
  char *s, *t;

  printf("a: 0x%x\n", (unsigned int) a);
  printf("b: 0x%x\n", (unsigned int) b);
  printf("c: 0x%x\n", (unsigned int) c);
  printf("\n");

  for (i = 0; i < 5; i++) printf("%12d ", b[i]);
  printf("\n");
  printf("\n");

  for (i = 0; i < 5; i++) printf("%s\n", c[i]);
  printf("\n");

  for (i = 0; i < 3; i++) {
    for (j = 0; j < 3; j++) {
      printf("%12d ", a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
```

```
  for (b = a[0]; b < (int *) a[0][0]; b += 2) {
    printf("%12d\n", *b);
  }
  printf("\n");

  /* Make this the last thing you do on the test.
     Don't burn time on it if you don't have the
     time to burn. */

  s = c[0];
  t = s+1;
  for (i = 0; i < 6; i++) {
    s[i] = *t;
    t += 7;
  }
  b = (int *) s;
  printf("%s 0x%x\n", s, *b);
}
```

The first three lines printed by **messy_proc()** are "a: 0xbfffdb30", "b: 0xbfffdb48" and "c: 0xbfffdb3c". Tell me what the rest of the output is. There are no segmentation violations or bus errors in this program (I have compiled and run it).

Question 4

Suppose your heap is composed of 384 bytes starting at address 0x1c230, pictured on the right. You are given the following assumptions:

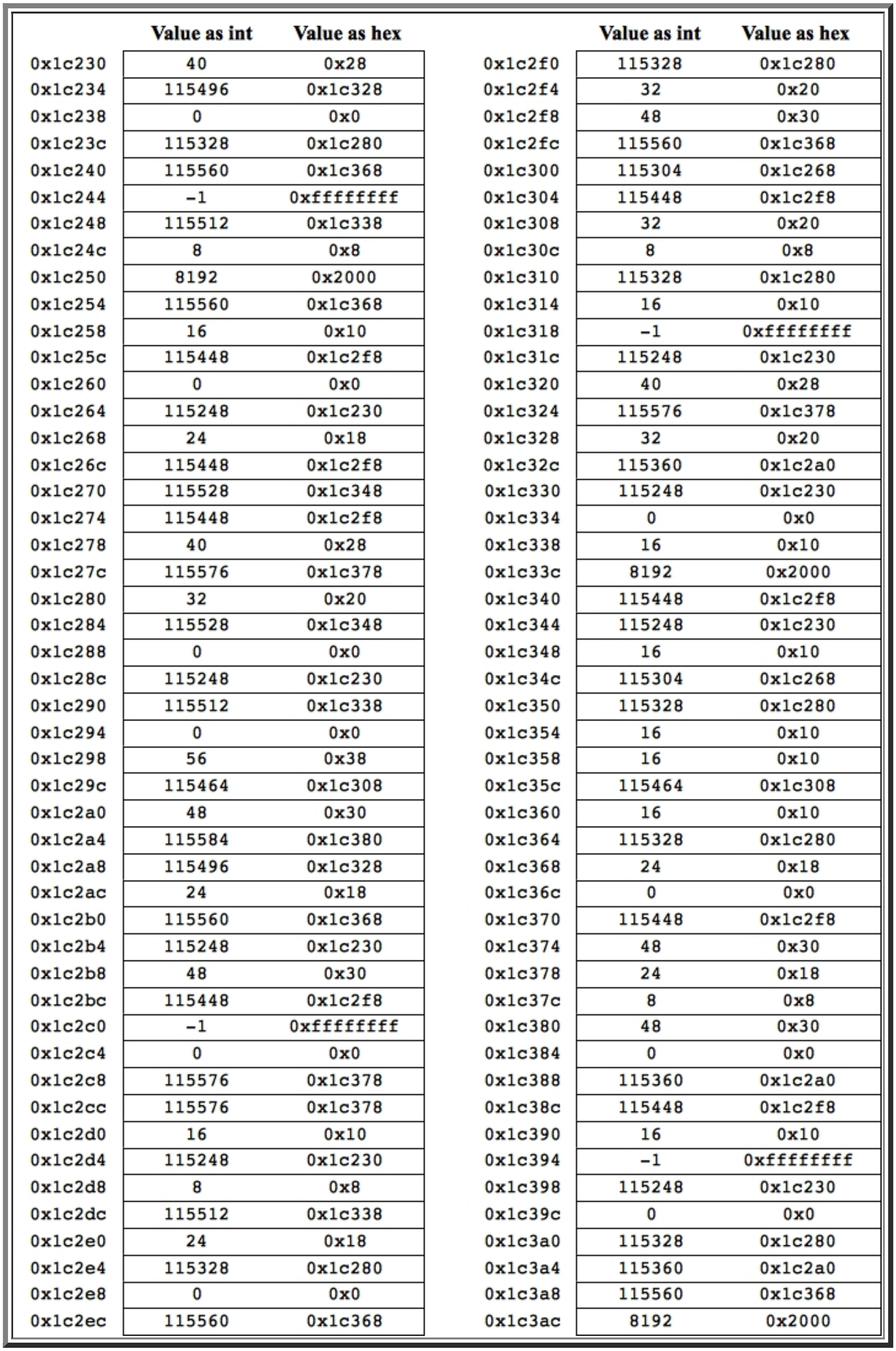
- Memory is allocated as described in class, where the size of an allocated block is stored eight bytes before the pointer.
- The free list starts at 0x1c280.
- Free list nodes contain size, flink and blink.
- Pointers are four bytes.

Part A: Tell me all of the nodes on the free list, in order. For each node, tell me the address of the node and its size.

Part B: Tell me all of the allocated chunks of memory. For each chunk, tell me the value that was returned from **malloc()**, and the total size of the chunk.

Part C: What would **sbrk(0)** return?

Part D: Suppose I have an integer pointer **j** whose value is 0x1c3c4. If I execute "**j = 55*", will the operation complete successfully, cause a segmentation violation or cause a bus error? Explain why.



	Value as int	Value as hex		Value as int	Value as hex
0x1c230	40	0x28	0x1c2f0	115328	0x1c280
0x1c234	115496	0x1c328	0x1c2f4	32	0x20
0x1c238	0	0x0	0x1c2f8	48	0x30
0x1c23c	115328	0x1c280	0x1c2fc	115560	0x1c368
0x1c240	115560	0x1c368	0x1c300	115304	0x1c268
0x1c244	-1	0xffffffff	0x1c304	115448	0x1c2f8
0x1c248	115512	0x1c338	0x1c308	32	0x20
0x1c24c	8	0x8	0x1c30c	8	0x8
0x1c250	8192	0x2000	0x1c310	115328	0x1c280
0x1c254	115560	0x1c368	0x1c314	16	0x10
0x1c258	16	0x10	0x1c318	-1	0xffffffff
0x1c25c	115448	0x1c2f8	0x1c31c	115248	0x1c230
0x1c260	0	0x0	0x1c320	40	0x28
0x1c264	115248	0x1c230	0x1c324	115576	0x1c378
0x1c268	24	0x18	0x1c328	32	0x20
0x1c26c	115448	0x1c2f8	0x1c32c	115360	0x1c2a0
0x1c270	115528	0x1c348	0x1c330	115248	0x1c230
0x1c274	115448	0x1c2f8	0x1c334	0	0x0
0x1c278	40	0x28	0x1c338	16	0x10
0x1c27c	115576	0x1c378	0x1c33c	8192	0x2000
0x1c280	32	0x20	0x1c340	115448	0x1c2f8
0x1c284	115528	0x1c348	0x1c344	115248	0x1c230
0x1c288	0	0x0	0x1c348	16	0x10
0x1c28c	115248	0x1c230	0x1c34c	115304	0x1c268
0x1c290	115512	0x1c338	0x1c350	115328	0x1c280
0x1c294	0	0x0	0x1c354	16	0x10
0x1c298	56	0x38	0x1c358	16	0x10
0x1c29c	115464	0x1c308	0x1c35c	115464	0x1c308
0x1c2a0	48	0x30	0x1c360	16	0x10
0x1c2a4	115584	0x1c380	0x1c364	115328	0x1c280
0x1c2a8	115496	0x1c328	0x1c368	24	0x18
0x1c2ac	24	0x18	0x1c36c	0	0x0
0x1c2b0	115560	0x1c368	0x1c370	115448	0x1c2f8
0x1c2b4	115248	0x1c230	0x1c374	48	0x30
0x1c2b8	48	0x30	0x1c378	24	0x18
0x1c2bc	115448	0x1c2f8	0x1c37c	8	0x8
0x1c2c0	-1	0xffffffff	0x1c380	48	0x30
0x1c2c4	0	0x0	0x1c384	0	0x0
0x1c2c8	115576	0x1c378	0x1c388	115360	0x1c2a0
0x1c2cc	115576	0x1c378	0x1c38c	115448	0x1c2f8
0x1c2d0	16	0x10	0x1c390	16	0x10
0x1c2d4	115248	0x1c230	0x1c394	-1	0xffffffff
0x1c2d8	8	0x8	0x1c398	115248	0x1c230
0x1c2dc	115512	0x1c338	0x1c39c	0	0x0
0x1c2e0	24	0x18	0x1c3a0	115328	0x1c280
0x1c2e4	115328	0x1c280	0x1c3a4	115360	0x1c2a0
0x1c2e8	0	0x0	0x1c3a8	115560	0x1c368
0x1c2ec	115560	0x1c368	0x1c3ac	8192	0x2000

Some useful prototypes

```
int strlen(char *s);  - Returns the length of a string

char *strcpy(char *dest, char *src);  - Copies the string in src to memory pointed to by dest.
                                      - Returns its first argument.

char *strdup(char *s);  - Allocates room for a copy of s, copies it and returns it.

char *strcat(char *dest, char *src);  - Assumes that dest is a string, and appends src to it.

char *strchr(char *s, char c)  - Returns a pointer to the first occurrence of c in s, or NULL.

char *strrchr(char *s, char c)  - Returns a pointer to the last occurrence of c in s, or NULL.

char *strstr(char *s, char *st)  - Returns a pointer to the first occurrence of st in s, or NULL.

int read(int fd, char *buf, int size);
```