IC221: Systems Programming
06-Week Written Exam
[SOLUTIONS]

February 12, 2014

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page. Show all work, but please be legible. *Microscopic writing will not be graded.*

You are allowed a single crib sheet for this exam on one-side of an 8.5"x11" sheet of paper, hand written. You must turn in your crib sheet with your exam.

Name: _____

Section: _____

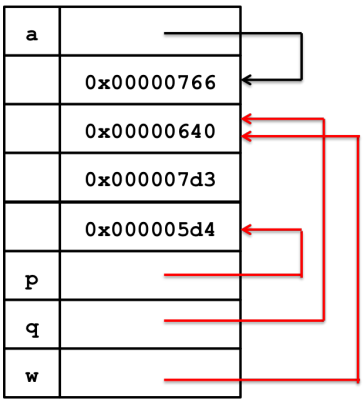
Alpha: _____

Question	Points	Score
1	20	
2	20	
3	20	
4	20	
5	20	
Total:	100	

1. Consider the following program:

```
int main(int argc, char * argv[]){

  //array of integers described in hex
  int a[4] = {0x00000766,  // (hint: two hex digits
              0x00000640,  //   represents one byte)
              0x000007d3,
              0x000005d4};


  int * p = a+3;
  int * q = p-2;

  char * w = (char *) &(a[1]); /* MARK */

  //print output as hex values with leading 0x
  printf("*p = 0x%x\n", *p);
  printf("q[1] = 0x%x\n", q[1]);

  printf("w[0] = 0x%x\n", w[0]);

  return 0;
}
```

(a) **[5 points]** Complete the memory diagram below for the pointers `p`, `q`, and `w`.



(b) **[3 points]** What is the output of the program for `*p` and `q[1]`:

`*p` = 0x000005d4

[+1 if right]

`q[1]` = 0x000007d3

[+1 if right (+3 for both right)]

(c) **[3 points]** Assume the address of the array `a` is at `0x0bffff04` what is the address of the second integer in the array, i.e., `&a[1]` or `a+1`? Explain.

`0x0bffff08` or 4 more than the address of `a` because `a` is an integer pointer, and adding 1 goes to the *next* integer, which is 4 bytes further. **[+2 for answer, +1 for explanation]**

(d) **[3 points]** Explain why the pointer cast at the line `MARK` is valid with respect to the data size of a pointer type? While the underlying data being referenced does not change, what does change when using a different pointer type?

Casting between pointers is valid because the pointer value does not change; however, it does change how we interpret the underlying data. Instead of treating the data is an integer, we now treat it as an array of characters. **[+2 for casting pointer; +1 for interpretation]**

(e) **[3 points]** What is the output for the `w[0]` output? *(Hint: it should be one byte or two hex digits.)*

`w[0]` = 0x40

[+3 if right, +1 for 0x00]

(f) **[3 points]** Which of the byte orderings, Big Endian or Little Endian, is used on most computers? Which has the *most significant* byte first, that is the byte most to the left, reading bytes left to right?

Little Endian is used on most computers, but Big Endian has the most significant byte first. **[+2 for part 1 and +1 for part 2]**

2. Consider the following `ls -al` output

```
aviv@saddleback: prob2 $ ls -al
total 32
drwxr-x--- 4 aviv scs  4096 Feb  6 10:19 .
drwx--x--x 5 aviv scs  4096 Feb  6 10:16 ..
drwxr-x--- 2 aviv scs  4096 Feb  6 10:17 bar
-rw-r----- 1 aviv mids    0 Feb  6 10:17 baz
drwxr-x--- 2 aviv scs  4096 Feb  6 10:17 foo
-rwxr----- 1 aviv mids 8516 Feb  6 10:18 helloworld
-rw-r----- 1 aviv scs   114 Feb  6 10:18 helloworld.c
```

(a) **[3 points]** What are the names of all the directories listed in the `ls -al` output?

> `.`, `..`, `bar`, `foo` **[+1 for `bar`, +1 for `foo`, +1 for `.` and `..`]**

(b) **[2 points]** What does `.` and `..` refer to?

> `.` refers to the current directory `..` refers to the parent directory **[+1 for each]**

(c) **[2 points]** Which users currently has access to *read* the file `baz`? Is there anything to read in this file?

> The owner has read and write access, the group has read access.
> The file is empty. **[+1 for access, +1 for empty]**

(d) **[3 points]** Provide a *single* command to change the permission of `baz` so that the owner has read, write, and execute permission, the group has read and execute, and everyone else just has execute permission.

> `chmod 751 helloworld` **[+1 for chmod, +1 for 751, +1 for helloworld]**

(e) **[5 points]** Consider the executable file *helloworld* which has group read permission but not execute? Explain how this does not properly secure the file from unauthorized execution by users in the `mids` group? How could the owner better protect the file?

> A user in the mids group can copy the file using their read permission, then they can set it to executable since they own the copy, and execute it. The owner should not allow group read permission. **[+3 for copy then execute, +2 for not allow group permission]**

(f) **[3 points]** Who has the privilege to change the permissions and group of a file/directory? Who has the privilege to change the owner of a file/directory?

> Owner can change the group or permissions, root/suerpuser can change owner **[+1 for owner and +1 for root/superuser, +3 for both]**

(g) **[2 points]** What command will update the time stamp on the file `baz` to the current time (right now, as you take this exam)?

> `touch` **[+2 for touch, +1 for `cat /dev/null > baz`]**

3. (a) **[3 points]** Consider the program below:

```
int main(int argc, char * argv[]){
  char str[12];

  if( argc < 2) //require a command
     return 1;  //line argument

  //copy the argument to str
  strcpy(str,argv[1]);

  //print it out
  printf("arg[1]: %s\n",str);

  return 0;
}
```

What is the *longest* command line argument, in terms of the number of characters, that will not *overflow* the character array `str` during the `strcpy`? Explain briefly?

The longest can be 11 characters long because the 12th must be NULL to be a string.
[+2 for 11, +1 for explain]

(b) **[3 points]** What string library function can you replace `strcpy()` with to ensure that `str` is not overflowed? Provide the function and its arguments below:

strncpy(str,arg[1],11)
[+1 for strncpy, +2 for 11 length, or +2 for 12 length] [+2 for strncpy with args in wrong order]

(c) **[4 points]** What is the output of the program to the right?

```
int main(int argc, char *argv[]){
  char s1[] = "Go Navy! Beat Army!";
  char *p = s1;

  printf("strlen(s1): %d\n",strlen(s1));
  printf("sizeof(s1): %d\n",sizeof(s1));

  printf(" strlen(p): %d\n",strlen(p));
  printf(" sizeof(p): %d\n",sizeof(p));
}
```

strlen(s1): 19
sizeof(s1): 20
strlen(p): 19
sizeof(p): 8
[+1 for each]

(d) **[3 points]** What is the difference between `sizeof()` and `strlen()`?

sizeof() measures how much memory is required , strlen() measures the length of the string up to the null terminator
[+1 for sizeof(), +1 for strlen(), +3 for both with good description]

(e) **[4 points]** Complete the function below which should return the equivalent of the `strlen()` function:

```
int my_strlen( char * string) {
  int len = 0;

  while ( *string++) len++;

  return len;
}
```

[+2 for recognize null termination, +1 for loop, +1 for increment len]

(f) **[3 points]** What does the following code snippet print:

```
char data[]="Don't\0 Panic!\0 And bring a towel!\0";
printf("%s\n",data+8);
```

"Panic" because "%s" will format until the null **[+1 for sizeof(), +1 for strlen()]**

4. For this question, assume you have a comma separated file (`database.csv`) with the following fields:

```
First Name,Last Name,Street Address,City,State,Phone Number
```

The state is a two-character postal code (e.g. MD for Maryland and CA for California). The phone number is always formatted like 999-999-9999. The `database.csv` is not corrupted, and none of the fields may have a comma in it.

(a) **[3 points]** Write a command line to count the number of unique states in `database.csv`. Your command should contain pipes and/or redirects.

```
cut -d "," -f 5 database.csv | sort | uniq | wc
```

[+2 for having all cut, sort, uniq, and wc, +1 for right arguments to cut]

(b) **[5 points]** What are the three guiding principles of the Unix design philosophy? Explain how the command above achieves these principles.

(1) Write programs that do one thing and do that one thing well
(2) Write program that work well together
(3) Write programs to handle text streams
[+1 for each principle]

(c) **[3 points]** Write a single command line to save all the phone numbers to a file called `phone.txt`. Your command may contain pipes and/or redirects.

```
cut -d "," -f 7 database.csv > phone.txt
```

[+1 for cut, +2 for redirection]

(d) **[3 points]** Suppose there is a second database file, `moredata.csv`, that has the same format. Write a command to append the phone numbers from `moredata.csv` to the end of the `phone.txt` file you created in the previous question. Your command may contain pipes and/or redirects.

```
cut -d "," -f 7 moredata.csv >> phone.txt
```

[+2 append redirection, +1 for cut]

(e) **[3 points]** Suppose you have a whole directory of database files formatted like above with names from `database.00.csv`, `database.01.csv`, `database.02.csv` and up to `database.99.csv`. Write a *single* glob with `wc` to count the number of lines in the database files number in the 10's, 30's, and 70's, but not database file 15, 35, or 75.

```
wc -l
```

```
database.[137][^5].csv
```

[+1 for using [ ], +1 for 137, +1 for not 5]

(f) **[3 points]** Consider now that you are currently in a directory that contains five sub-directories full of database files numbered between 00 and 99, like above. Each the directories have the following naming scheme: `data_2015`, `data_2016` `data.2017`, `data.2018`. Write a *single* glob with `wc` to count the number of lines in all the databases found in year 2015 and 2017 sub-directories? *(Note: the separators _ and . are used in the sub-directory names)*

```
wc -l
```

```
data?201[57]/*
```

[+2 for ? +1 for [57]]

5. The questions refers to the program below:

```
typedef struct{
  int left;
  int right;
} pair_t;

int main(int argc, char * argv[]){
  int num, i;

  if ( argc < 2 ){
    fprintf(stderr,
            "%s: ERROR: require number of pairs\n",
            argv[0]);
  }

  num = atoi(argv[1]);

  /* MARK 1*/
  pair_t * pairs[num];

  printf("Enter space separated pairs\n");
  for( i=0; i<num; i++ ){
    pairs[i] = malloc(sizeof(pair_t);
    scanf("%d %d", &(pairs[i]->left)),
                   &(pairs[i]->right));
  }
  /* END MARK 1*/

  for( i=0; i<num;i++){
   printf("(%d,%d)\n", pairs[i]->left,
                       pairs[i]->right);
  }

  /* MARK 2 */
  return 0;
}
```

(a) **[3 points]** In the program to the left, **circle** the command(s) that write to standard output, **square** the command(s) that read from standard input, and **underline** the command(s) that write to standard error.

(b) **[3 points]** Suppose the program is compiled to `pair-reader` and executed like such, without any additional arguments, what is the output of the program?

```
./pair-reader
```

> `./pair-reader: ERROR: Require number of pairs` **[+2 for ERROR line, +1 for ./pair-reader]**

(c) **[2 points]** What is the output of the program when executed as follows:

```
echo "1 8 5 4" | ./pair-reader 2
```

> `(1,8)`
> `(5,4)`
> **[+1 for each line]**

(d) **[5 points]** The program has a memory leak, describe the memory leak below, and provide the code to be inserted at `MARK 2` that will properly deallocate all the memory:

> the malloc()'d pairs are not free()'ed
>
> `for(i=0;i<num;i++){ free(pairs[i]); }`
>
> **[+2 for explanation, +3 for code]**

(e) **[3 points]** Consider replacing the `MARK 1` portion of the program with the following function call:

```
pairs = get_pairs(num);
```

where `get_pairs()` is defined as follows

```
pairs_t * get_pairs(int num){
   pair_t * pairs[num]; /* MARK 3*/

   printf("Enter space separated pairs\n");
   for( i=0; i<num; i++ ){
    pairs[i] = malloc(sizeof(pair_t);
    scanf("%d %d", &(pairs[i]->left)),
                   &(pairs[i]->right));
  }

  return pairs;
}
```

Describe what is *wrong* with this modification to the program with respect to the program memory layout? Be sure to mention stack and heap.

> The `pairs` array is allocated on the stack in the function, after the function returns it is deallocated, while data allocated on the heap persist. **[+2 stack allocation, +1 heap description]**

(f) **[4 points]** Rewrite the line of code at `MARK 3` to fix the error:

> `pair_t ** pairs = calloc(num, sizeof(pair_t *))` **[+2 for right type for pairs, +1 for calloc/malloc, +1 for sizeof a pointer to a pair]**