IC221: Systems Programming
6-Week Practicum

February 14, 2014

Instructions:

- This is a 1 hour and 50 minute practicum.
- The practicum is open notes, open book, and open internet. Previous lab submissions count as notes. You may not, however, communicate with anyone directly, e.g., via intstant message.
- Complete all the programs as described in this document and in the source code provided.
- To retrieve the source, run `~aviv/bin/ic221-up`, then `cd ~/ic221/practicum/6-week`
- All code submissions will occur via the `~aviv/bin/ic221-submit` with the option `practicum/6-week`. You can submit multiple times, only your final submission will be graded.
- You must also turn in this document at the completion of the examination. Feel free to include any notes or comments in writing within this document.
- You can **test your submission** by running the test script `test.sh` from the `~/ic221/practicum/6-week` directory. The test script will report pass or fail for each grading unit.

Name: _____

Alpha: _____

Question	Points	Score
1	5	
2	10	
3	10	
Total:	25	

1. [**5 points**] Change into the `ic221/practicum/6-week/bash` directory, where you will find the following files:

   - `allnumbers.sh` : Bash script you are to complete
   - `isnumber` : Compiled executable
   - `isnumber.c` : Source code for `isnumber`

   Your goal is to complete the bash script `allnumber.sh` which will filter out all arguments that are numbers, and echo them to the terminal line by line. For example:

```
#> ./allnumbers.sh 1 b 2 3 a 5 -12
1
2
3
5
-12
#> ./allnumbers.sh a b c
#> ./allnumbers.sh
#> ./allnumbers.sh 10 a 3
10
3
```

   To assist in this process, you are provided an executable `isnumber`, which reads a "word" from standard input, and exits with status 0 if the word is a number, or status 1 if the word is *not* a number. Here is an example of its usage:

```
#> echo "12" | ./isnumber ; echo $?
0
#> echo "a" | ./isnumber ; echo $?
1
```

   The source code for `isnumber` is provided below and is in the `bash` directory.

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

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

  //Try a read a number from *stdin*
  if(scanf("%d", &i) == 1){
    return 0; //exit succes, 0, if is a number
  }else{
    return 1; //exit failur, 1, if not a number
  }

}
```

2. [**10 points**] Change into the `ic221/practicum/c-prog` directory, where you will find the following files:

   - wordcounter.c : Source Code you are to complete
   - wordcounter : Compiled executable for the source code
   - sample.txt : Small test file
   - hemmingway.txt : Longer test file

   Your goal is to complete the `wordcoutner.c` program, which will is compiled to `wordcounter` using the following command:

```
#> clang -g wordcounter.c -o wordcounter
```

   The `wordcounter` program read from a file, stores each word and its length in an array of structures, printing the list of words and their length once finished. The name of the files to read from are provided as a command line argument. The structures used for storing words are as follows, and are found in `wordcounter.c`:

```
typedef struct{

  char * str; //pointer to a string
  int len;    //length of the string

} word_t;

typedef struct{

  word_t ** list; //array of word_t pointer
  int size;       //size of the array
  int nwords;     //number of words in the array

} wordlist_t;
```

   The function `loadlist()` is completed for you, as well as the `main()` function. You should review the provided code for details on how the structures are used.

   You must complete the `printlist()` and `deletelist()` function. The `deletelist()` functions should properly deallocate a `wordlist_t` so that there are no memory leaks. You should check for memory leaks using `valgrind`.

   The `printlist()` function will iterate through the `list` of a `wordlist_t` printing out information about each `word_t` using the following print statement, where `i` is the index in `list`.

```
  printf("%d: %s %d\n", i, wordlist->list[i]->str, wordlist->list[i]->len);
```

   You **must** use the format output above to receive full credit. Error checking and reporting for file I/O is provided for you, so the only additional output should be the print statements above.

3. [**10 points**] Change into the `ic221/practicum/6-week/sys-prog` directory, where you will find the following files:

   - `read-struct.c` : Program you are to complete
   - `read-struct` : Compiled executable
   - `simple-db` : Simple database file for testing
   - `bank-db` : Larger database file for testing

   Your goal is to complete the `read-struct.c` program so that it can read the contents of the `bank-db` and `simple-db` databases. To complete this task, you must use the `read()`, `open()`, and `close()` system call. `read-struct.c` should be compiled using the following command:

```
#> clang -g read-struct.c -o read-struct
```

   The database files contain the raw bytes of the following structure, defined in `read-struct.c`:

```
struct bank_entry{
  char first_name[64]; //string to store first name, max length 63+1 for NULL
  char last_name[64]; //string to store last name, max lenght 63+1 for NULL
  unsigned int balance; //account balance
  unsigned int credit; //allowable credit
};
```

   The `read-struct.c` program, given a file name as an argument, opens that file and attempts to read all the data in that file to produce the following output:

```
#> ./read-struct simple-db
Name: Aviv, Adam
Bal: 1000
Crd: 20

#> ./read-struct bank-db
Name: Aviv, Adam
Bal: 2000
Crd: 10
Name: Blenkhorn, Kevin
Bal: 11231
Crd: 54
Name: McDowell, Luke
Bal: 5234
Crd: 10
Name: Chambers, Nate
Bal: 0
Crd: 8755
Name: Sikora, Matt
Bal: 22
Crd: 3872
Name: Watt, Eric
Bal: 42
Crd: 42
```

   Each bank entry structure **must** be outputted using the following format print to receive full credit:

```
printf("Name: %s, %s\n", be.last_name, be.first_name);
printf("Bal: %d\n", be.balance);
printf("Crd: %d\n", be.credit);
```

   If an error occurs for any of the system calls, your program should report the error by printing to **standard error**.