University of Maryland College Park
Department of Computer Science
CMSC216 Spring 2017
Midterm II

Last Name (PRINT): _____

First Name (PRINT): _____

University Directory ID (e.g., umcpturtle) _____

Lab TA (Circle One):

0101 Shravan Sanjiv, 8 am	0201 Adam Hamlin, 12 pm	0301 Raghav Bhasin, 12 pm	0305 Stefani Moore, 8 am
0102 Stefani Moore, 9 am	0202 Allen Cheng, 1 pm	0302 Raghav Bhasin, 1 pm	0306 Victor Chang, 1 pm
0103 Lee Williams, 10 am	0203 Shravan Sanjiv, 3 pm	0303 Gabriella Farley, 3 pm	0307 Gabriella Farley, 4 pm
0104 Adam Hamlin, 11 am	0204 Khanh Nguyen, 4 pm	0304 Justin Park, 4 pm	0308 Stefan Su, 5 pm
Honors			

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: _____

Instructions

- If the answer to a question depends on the architecture involved, assume the question applies to behavior on linux.grace.umd.edu.
- This exam is a closed-book and closed-notes exam.
- Total point value is 200 points.
- The exam is a 75 minutes exam.
- Please use a pencil to complete the exam.
- Your code must be efficient.
- **You don't need to use meaningful variable names; however, we expect good indentation.**
- MACHINE PARAMETERS: sizeof(char) == 1; sizeof(int) == 4; sizeof(long) == 8; sizeof(void *) == 8;

Grader Use Only

#1	Problem #1 (Miscellaneous)	(30)	
#2	Problem #2 (File I/O)	(30)	
#3	Problem #3 (Linked Lists)	(90)	
#4	Problem #4 (Assembly)	(50)	
Total	Total	(200)	

Problem #1 (Miscellaneous)

1. (3 pts) Write a Unix command that will create a compressed gzip archive file with the contents of the directory **project3** that is in the current directory. You can use any name for the archive file.

2. (9 pts) What is the output of the following program?

```
#include <stdio.h>
int main() {
   unsigned char m = 0x6b, p = 0xc3;

   printf("m | p: %02x\n", m | p);
   printf("m ^ p: %02x\n", m ^ p);
   printf("m >> 3: %02x\n", m >> 3);

   return 0;
}
```

3. (8 pts) The file **lamp.c** includes the file **switch.h,** and the file **switch.h** includes the file **voltage.h**. In addition, the file **switch.c** includes **switch.h**. Write one or more makefile rules that generate the file **lamp.o**. You do not need to use any macros and the only compilation option is -Wall.

4. (10 pts) The following program will be used for the questions below.

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

char val = 'c';

int main() {
   char *p = &val, *a = NULL, b[] = "Hi";
   char *m = malloc(2), *x = m;

   /* CODE HERE */

   return 0;
}
```

Which of the following are valid (will compile and run without errors) if they were to replace the comment **/* CODE HERE */** above? Circle either VALID or INVALID.

a. free(p); /* VALID or INVALID */
b. free(a); /* VALID or INVALID */
c. free(b); /* VALID or INVALID */
d. free(b + 2); /* VALID or INVALID */
e. a = b; /* VALID or INVALID */
f. b = a; /* VALID or INVALID */
g. b[0] = 'L'; free(x); *m = *b; /* VALID or INVALID */
h. free(m + 1); /* VALID or INVALID */
i. free(m); m = b; /* VALID or INVALID */
j. printf("%c", *a); /* VALID or INVALID */

Problem #2 (File I/O)

Define the function **load_data** that has the prototype shown below. The function reads information (id, gpa, and name) about a single student from a file. The first line of the file is a header and the second line has the student's information. The specifications for this problem are:

- The format of the student's information is id, gpa, and name separated by *. Any number of spaces can appear between each item. You can assume only the first name of a student will be provided.
- The function will initialize the out parameters id, gpa, and name with the corresponding information.
- You can assume the file will always have a header and a second line with student's information. The function will read and throw away the header.
- The function will return 0 if any parameter is NULL or if opening of the file fails. No error message (regarding file opening failure) will be generated.
- The function returns 1 if the data is read.
- You can assume lines will have a maximum of 80 characters.
- Below you will find an example of using the function you are expected to write. Feel free to ignore the example if you know what to implement.
- Provide your answer on the next page.

```
int load_data(const char *filename, int *id, float *gpa, char *name)
```

```
#define MAX_LINE 80

int main() {
   char name[MAX_LINE + 1];
   int id;
   float gpa;

   load_data("data.txt", &id, &gpa, name);
   printf("values: %d %f %s\n", id, gpa, name);

   return 0;
}
```

```
% cat data.txt
# Name*id*gpa
101 * 3.45 *   John
% a.out
values: 101 3.450000 John
%
```

Standard IO Cheat Sheet

- FILE *fopen(const char *path, const char *mode);
- int fscanf(FILE *stream, const char *format, ...);
- char *fgets(char *s, int size, FILE *stream);
- int sscanf(const char *str, const char *format, ...);
- int fputs(const char *s, FILE *stream);
- int fprintf(FILE *stream, const char *format, ...);
- int fclose(FILE *fp);

PAGE FOR YOUR CODE

```
int load_data(const char *filename, int *id, float *gpa, char *name) {
```

Problem #3 (Linked Lists)

The following structures and definitions will be used for the questions below. A linked list is used to keep track of scores for students in a course. A **Linked_list** structure keeps track of the actual head of the list and its size. **For this problem you can assume memory allocations are always successful (do not generate an error message if allocation fails).** Take a look at the driver and expected output we have provided below, if you have any questions regarding the functionality you need to implement. You can ignore the driver if you know what to implement. **The driver relies on functions (create_list, print_list, destroy_list, add) that you do not need to implement and that you MAY not use for the implementation of any of the functions below.**

```
#define MAX_LEN 80

typedef struct student {
   char name[MAX_LEN + 1];
   int score;
} Student;
```

```
typedef struct node {
   Student *student;
   struct node *next;
} Node;
```

```
typedef struct list {
   Node *head;
   int size;
} Linked_list;
```

1. Define the function **duplicate_node** that has the prototype shown below. The function makes a copy of the node associated with the **src_node** parameter by creating a new node and a new **Student** structure. If **set_next_null** is true, the **next** pointer of the new node will be set to NULL; otherwise it will be set to the value associated with the **next** field of the **src_node** parameter. You can assume **src_node** is different than NULL.

```
Node *duplicate_node(Node *src_node, int set_next_null)
```

2. Define the function **remove_first** that has the prototype shown below. The function removes the first element from the list (if any). You must deallocate any dynamically-allocated memory associated with the element. If the list is **initially** empty, the memory associated with the **Linked_list** structure must be freed and the pointer variable associated with the **list** parameter must be set to NULL. If initially the list has at least a single element, the **Linked_list** structure will NOT be freed. You can assume the **list** parameter is different than NULL.

```
void remove_first(Linked_list **list)
```

3. Define the function **get_filtered_list** that has the prototype shown below. The function returns a new list with **copies** of elements from the **list** parameter that satisfy the filter condition defined by the **filter_func** parameter. The **filter_func** function will return true if a student satisfies the filter condition, and false otherwise. An empty list (instead of NULL) will be returned if no element from the **list** satisfies the filter condition. Use the **duplicate_node** function defined above for this problem, even if you did not implement it. You can assume all parameters are different that NULL.

```
Linked_list* get_filtered_list(Linked_list *list, int (*filter_func)(Student *student))
```

```
int filter_by_score(Student *student) { return student->score <= 20 ? 1 : 0; }
int main() {
   Linked_list *my_list, *filtered_list;
   const char *names[] = {"Mike", "Rose", "Mike"};
   int scores[] = {80, 4, 19}, i;
   create_list(&my_list, NULL, 0);
   for (i = 0; i < 3; i++) { add(my_list, names[i], scores[i]); }
   print_list(my_list);
   filtered_list = get_filtered_list(my_list, filter_by_score);
   print_list(filtered_list);
   destroy_list(my_list);
   destroy_list(filtered_list);
   create_list(&my_list, NULL, 0);
   add(my_list, "Tom", 100);
   print_list(my_list);
   remove_first(&my_list); /* my_list != NULL after the function call */
   print_list(my_list);
   remove_first(&my_list); /* my_list == NULL after the function call */
   return 0;
}
```

```
% a.out
List size: 3
Name: Mike, Score: 19
Name: Mike, Score: 80
Name: Rose, Score: 4
List size: 2
Name: Mike, Score: 19
Name: Rose, Score: 4
List size: 1
Name: Tom, Score: 100
Empty list
%
```

PAGE FOR YOUR CODE

PAGE FOR YOUR CODE

PAGE FOR YOUR CODE

Problem #4 (Assembly)

Complete the assembly function named **chg_to_even** you will find on the next page. The function corresponds to the C function you see below. The function updates values that are odd to even by adding 1. The function returns the number of values that were updated.

- You only need to provide assembly code for the **chg_to_even** function.
- Do not provide assembly code for main nor the print_array function.
- On the next page you will see assembly representing the array.
- Your assembly code must work for any array (not just the one in the example).
- **Try to provide comments with your assembly code. It will help us understand what you are trying to do.**
- **Your code must be efficient.**
- The function must save and restore the base pointer.
- Parameters must be passed on the stack.
- You may not modify the provided function and implement the modified version.
- **You need to define and use the local variable (cnt).** That is, at the beginning of the **chg_to_even** function you need to reserve space on the stack for this local variable and you must use the space.
- **Your solution must be recursive otherwise you will lose most of the credit for this problem.**
- Use **%ebx** to represent the array **data** parameter and **%ecx** the **length** parameter.

```
#include <stdio.h>

int chg_to_even(int *data, int length) {
   int cnt = 0;

   if (length == 0) {
      return 0;
   } else {
      if (*data % 2 != 0) {
         (*data)++;
         cnt = 1;
      }
   }

   return chg_to_even(data + 1, length - 1) + cnt;
}

int main() {
   int src[] = { 1, 20, 5, 30, 17 };
   int length = 5;

   printf("Total: %d\n", chg_to_even(src, length));
   print_array(src, length);

   return 0;
}
```

Output

```
Total: 3
2 20 6 30 18
```

Assembly Cheat Sheet

Registers: %eax, %ecx, %edx, %ebx, %esi, %edi, %esp, %ebp
Assembler Directives: .align, .long, .pos
Data movement: irmovl, rrmovl, rmmovl, mrmovl
Integer instructions: addl, subl, multl, divl, modl
Branch instructions: jmp, jle, jl, je, jne, jge, jg
Reading/Writing instructions: rdch, rdint, wrch, wrint
Other: pushl, popl, call, ret, halt
Ascii code for newline character: 0x0a
Ascii code for space: 0x20

```
chg_to_even:     pushl %ebp
                 rrmovl %esp, %ebp

                 mrmovl 8(%ebp), %ebx  # Retrieving array pointer
                 mrmovl 12(%ebp), %ecx # Retrieving length
```

```
func_end:        rrmovl %ebp, %esp
                 popl %ebp
                 ret

.align 4
data:      .long 1
           .long 20
           .long 5
           .long 30
           .long 17
```

EXTRA PAGE

EXTRA PAGE