University of Maryland College Park
Department of Computer Science
CMSC216 Spring 2018
Midterm II

Last Name (**PRINT IN UPPERCASE**) _____

First Name (**PRINT IN UPPERCASE**): _____

University Directory ID (e.g., **your grace login id**) _____

Lab TA (Circle One):

0102 9am, Drake Petersen	0203 3pm, Stefan Su	0304 4pm, Jordan Wolinsky
0103 10am, Drake Petersen	0204 4pm, Michael Shultz	0305 8am, Michael Shultz
0104 11am, Ivan Quiles	0301 12pm, Raghav Bhasin	0306 1pm, Angel Wen
0201 12pm, Ivan Quiles	0302 1pm, Shyam Pujara	0307 4pm, Shyam Pujara
0202 1pm, Stefan Su	0303 3pm, Isaac Lee	0308 5pm, Yuval Reiss

Instructions

- Please print your answers and use a pencil.
- To make sure Gradescope can recognize your exam, print your name, write your grace login id at the bottom of each page, provide answers in the rectangular areas provided, and do not remove any exam pages. Even if you use extra pages for scratch work, they must be returned with the rest of the exam.
- Circle the correct lab session. You will lose (2 pts) if you circle an incorrect section.
- This exam is a closed-book, closed-notes exam, with a duration of 75 minutes and 200 total points.
- 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)	(40)	
#2	Problem #2 (Linked Lists)	(90)	
#3	Problem #3 (Assembly)	(70)	
Total	Total	(200)	

Problem #1 (Miscellaneous)

1. (3 pts) Declare a function pointer named **task** that allows us to complete the assignment below.

```
double *process(int *p, float y) {
   return NULL;
}

int main() {
   /* Declare variable here */
      double *(*task)(int*, float);

   task = process;

   return 0;
}
```

2. (3 pts) Define a Unix command that will uncompress a gzip archive file named **my_examples.tar.gz**.

3. (3 pts) Which of the following will allow the file **data.x** to be read and executed only by the owner? Circle all that apply.

r | w | x

   a. chmod 405 read.x
   b. chmod 555 read.x
   c. chmod 500 read.x
   d. chmod 650 read.x
   e. None of the above.

4. (3 pts) Which of the following expressions when assigned to val (the expression will replace EXPRESSION_HERE) will allow us to print 20.000000? Circle all that apply.

```
int x = 20, *p = &x;
float val = EXPRESSION_HERE;
printf("%f\n", val);
```

   a. *(float *)p
   b. (float)*p
   c. (float)&p
   d. None of the above.

5. (6 pts) The file **table.c** includes the files **bucket.h** and **table.h**. Complete the following makefile rule associated with the file **table.o**. You don't need to use any macros and the necessary flags are –g and any other flags you understand are needed.

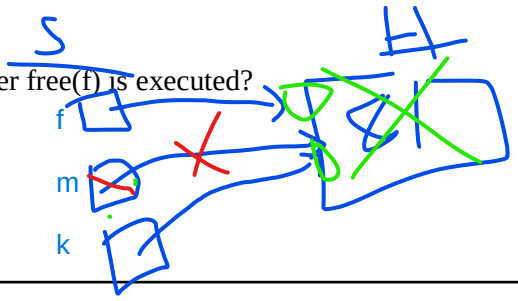
```
table.o: table.c table.h bucket.h
    gcc -g -c table.c
```

GraceLoginId:

6. (3 pts) How many dangling pointer variables do we have after free(f) is executed?

```
char *f = malloc(81);
char *m = f, *k = f;

m = NULL;
free(f);
```



Number of dangling pointer variables: 2

7. (3 pts) Define a Unix command using grep that lists the lines(s) in the file(s) ending with a .c extension containing the word **TODO**.

grep TODO *.c

8. (8 pts) Complete the body of the **flip_nibbles** function below. A nibble is a set of 4 bits. The function flips the position of nibbles in a byte. For example, if the function is called with 0xab, the function will return 0xba. **You may not use loops**.

```
unsigned char flip_nibbles(unsigned char val) {
        return (val >> 4) | (val << 4)
```

9. (8 pts) Complete the body of the **is_little_endian** function below. The function returns true if the environment stores data in little endian order and false otherwise. You must use the variable **value** to determine whether the environment is little endian or not. **You may not use loops**.

```
int int is_little_endian() {
   unsigned int value = 0x01234567;
```

GraceLoginId:

Problem #2 (Linked Lists)

The following structures and definitions are associated with a book abstraction. A book is represented by a linked list of **Page** structures (each Page structure is dynamically allocated). The head of the list is represented by the **Book** structure field **first_page**. Each page keeps track of the page contents by using a dynamically allocated string (**contents** field). A book with no pages is represented by a **Book** structure with **first_page** set to NULL. We have provided a driver (that you can ignore) in order to clarify the functionality you are expected to implement. Read the problem in its entirety before you start implementing any function.

```
#define MAX_LEN 80

typedef struct page {
   char *contents;
   struct page *next;
} Page;
```

```
typedef struct list {
   char title[MAX_LEN + 1];
   int num_pages;
   Page *first_page;
} Book
```

For this problem:

- **You can assume memory allocations are always successful (there is no need to add a conditional to check whether the allocation succeeded).**
- The driver relies on several functions (e.g., destroy_book) that you do not need to implement and that you MAY NOT use during the implementation of any of the functions below.
- You may not add any additional functions unless specified otherwise.

GraceLoginId:

1. (45 pts) Define the function **add_page_after** that has the prototype shown below. The function adds a page after the page associated with the **num** parameter. For example, to add a page after the second page, **num** will have a value of 2. If the value of **num** is 0, the page will become the first page in the book. The function must store a copy of the **contents** parameter. For this problem you can assume the **book** and **contents** parameters will not be NULL and **num** is valid (a value in the range [0, number of book pages]).

```
void add_page_after(Book *book, int num, const char *contents){

    Page *curr = book->first_page, *prev = NULL, *new_page;
    int i = 0;
    while(i++ <= num){
        if(curr != NULL){
            prev = curr;
            curr = curr->next;
        } else {
            break;
        }
    }

    new_page = (Page*)malloc(sizeof(Page));
    new_page->contents = (char*)malloc(strlen(contents)+1);

    if(curr =){

    }
```

GraceLoginId:

2. (45 pts) Define the function **merge_page** that has the prototype shown below. The function merges the target page (the one associated with **num**) with the page that follows (if any). If no page follows the target page, the function will not perform any processing. To merge two pages, replace the contents of the target page with the concatenation (appending) of its contents string, a newline character, and the contents string of the page that follows. After the concatenation, the next page must be removed from the book **(make sure you free any memory that was dynamically-allocated**). The function will return 0 if no merging took place and 1 otherwise. For this problem:

    a. **You will lose significant credit if:**
        i. You do not provide a recursive solution.
        ii. You add more than one auxiliary function.
        iii. You use realloc.
    b. You can assume **book** is not NULL and **num** is valid (a value in the range [1, number of book pages]).

```
void merge_page(Book *book, int num){
```

GraceLoginId:

ADDITIONAL PAGE IN CASE YOU NEED IT

GraceLoginId:

Sample Driver and Associated Output
Feel free to ignore. Return this page with the exam.

```
int main() {
   Book *book;

   create_book(&book, "Biography");
   add_page_after(book, 1, "In this class\nwe learn C.");
   add_page_after(book, 2, "In addition we learn\nUnix and emacs.");
   add_page_after(book, 3, "There are at least\n7 projects.");
   print_book(book);

   printf("***** After merge\n");
   merge_page(book, 2);
   print_book(book);

   print_to_file(book, "book.txt");

   destroy_book(book);

   return 0;
}
```

Output

```
Title: Biography
Number Pages: 3
Page #1:
In this class
we learn C.
Page #2:
In addition we learn
Unix and emacs.
Page #3:
There are at least
7 projects.
***** After merge
Title: Biography
Number Pages: 2
Page #1:
In this class
we learn C.
Page #2:
In addition we learn
Unix and emacs.
There are at least
7 projects.
```

Contents of data.txt file (created by print_book)

```
In this class
we learn C.In addition we learn
Unix and emacs.
There are at least
7 projects.
```

Standard I/O 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);`

GraceLoginId:

Problem #3 (Assembly)

Complete the assembly function named **encode** you will find on the next page. The function corresponds to the C function you see below. For this problem:

- You only need to provide assembly code for the **encode** function.
- Provide comments with your assembly code; they will be worth at least (6 pts).
- Your code must be efficient.
- The function assumes the parameter is provided via r25:24 and the result is returned in r25:24.
- You may not modify the provided function and implement the modified version.
- **Your solution must be recursive otherwise you will lose most of the credit for this problem.**

```
uint16_t encode(char *a) {
   if (*a != '\0')  {
      *a += 10;
      return 1 + encode(a + 1);
   }

   return 0;
}
```

Assembly Cheat Sheet

- Assembler Directives
  - .set
  - .text
  - .data
- Registers
  - r0 → freely available register that can be used for temporary values
  - r1 → assume to always hold value of 0; must be cleared if used
  - Caller-saved: r18 → r27, r30 → r31
  - Callee-saved: r2 → r17, r28 → r29
  - X → r27:r26
  - Y → r29:r28
  - Z → r31:r30
- Instructions (K and q represent constants)

ldi Ra, K	lds Ra, address	sts address, Rs	clr Ra
add Ra, Rb	adc Ra, Rb	adiw Ra, K	sub Ra, Rb
subi Ra, K	sbiw Ra, K	inc Ra	dec Ra
mul Ra, Rb	movw Ra, Rb	lsl Ra	lsr Ra
push Ra	pop Ra	call address	mov Ra, Rb
breq address	brne address	brge address	brlt address
brlo address	brsh address	cpi Ra, Rb	cpi Ra K
tst Ra	ld Ra, X / Y / Z	ld Ra, X+ / Y+ / Z+	ld Ra, -X, -Y, -Z
ldd Ra, Y + q	ldd Ra, Z + q	nop	

GraceLoginId:

```
    .global encode;
encode:
```

GraceLoginId:

Additional Page (for any problem). Make a note for which problem, and return with exam.

GraceLoginId:

Additional Page (for any problem). Make a note for which problem, and return with exam.

GraceLoginId: