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

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):_____

STUDENT ID (e.g. 123456789): _____

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

Your signature: _____

Instructions

- Please print your answers and use a pencil.
- Do not remove the staple from the exam. This will interfere with the Gradescope scanning process.
- 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 the provided extra pages for scratch work, they must be returned with the rest of the exam.
- 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.
- Cheat sheets can be found on page #2.
- 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 (I/O)	(40)	
#3	Problem #3 (Linked Lists)	(80)	
#4	Problem #4 (Assembly)	(50)	
Total	Total	(200)	

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);`

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	cp 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:

Problem #1 (Miscellaneous)

1. (3 pts) Define a Unix command that will move all the files that end with .pptx extension to the folder named **pre** found in your home directory.

```
mv *.pptx ~/pre
```

2. (3 pts) Using the Unix chmod command, define the file permissions for a file named **sym** as follows:
   a. The file can be written, read and executed by the owner.
   b. Users other than the owner can only read and execute the file.

```
chmod u=rwx go=rx sym
```

3. (4 pts) The following code is correct, but it has some unnecessary code. Indicate which parts can be removed.

```
char *f() {
   char *p = (char *)malloc(sizeof(char) + 3);
   if (p)
     return strcpy(p, "No");
   return NULL;
}

int main() {
   char *m = f();
   printf("%s\n", m);
   if (m != NULL)
      free(m);

   return 0;
}
```

1) casting malloc as char*
2) sizeof(char)+3 can be replaced with just 4
3) if(m != NULL) in main is pointless, NULL can be freed (equiv to malloc(0))

4. (12 pts) Write a makefile for the following code that will generate an executable named **car**. You don't need to use any macros and you don't need the –g flag. Remember that you must avoid unnecessary compilations. The unix cat command displays the contents of a file.

```
% cat car.c
#include <stdio.h>
#include "eng.h"

int main() {
  start();

  return 0;
}
% cat eng.c
#include <stdio.h>
#include "eng.h"

void start() { printf("On\n"); }
% cat eng.h
void start();
```

```
car: car.o eng.o
gcc -c car.o eng.o -o car

car.o: car.c eng.h
gcc -c car.c

eng.o: eng.c eng.h
gcc -c eng.c
```

GraceLoginId:

5. (8 pts) Implement the function **remove_middle_byte** that returns a value where the middle byte of the parameter has been removed. For example, the function call **remove_middle_byte(0xfabc)** will return **0xfc**. You may not use loops.

```
uint16_t remove_middle_byte(uint16_t value) {

        return (value & 0x000f) | ((value >> 12) << 4);



        /*
        1) value & 0x000f will retrieve last 4 bits of value because f is 1111, so any 1 & 1 = 1 and 1 & 0 = 0
        so we get 0x000(last 4 bit value)
        2) (value >> 12) << 4 will bring first 4 bits of value to second hex position by shifting 12 right and 4 back
        so we get 0x00(first 4 bit value)0
        */
```

GraceLoginId:

Problem #2 (I/O)

The **gen_histogram** function generates a histogram based on integer values provided in a file. For this problem:

1. The file consists of one number per line.
2. If the first word in a line is IGNORE, the line will be ignored.
3. If the parameter is NULL, standard input will be used.
4. All the output is sent to standard output.
5. You can assume all system calls are successful.
6. Each line has a maximum of 80 characters.
7. Below we provide an example of calling the function with the file data.txt.

```
% cat data.txt
5
 IGNORE  20
7
4
```

```
*****
*******
****
```

```
void gen_histogram(const char *file)

    FILE *input;
    char line[MAX_LEN +1], str[MAX_LEN+1];

    if(file == NULL)
       input= stdin;
    else
       input = fopen(file, "r");

    while(fgets(line, MAX_LEN+1, input)){
       int i, star_count;
       sscanf(line, "%s", str);
       if(strcmp(str, "IGNORE")){
          sscanf(line, "%d", &star_count);
          for(i = 0; i < star_count; i++){
             fprintf(stdout, "*");
          }
          fprintf(stdout, "\n");
       }
    }
```

GraceLoginId:

Problem #3 (Linked Lists)

The following structures and definitions are associated with a linked list of students. A **Student** structure keeps track of a student's name, student's id and a pointer to data (**data** field) associated with the student. The **List** structure keeps track of a function (**free_func**) which, if not NULL, will be used to free the dynamically-allocated memory associated with the **data** field of a student. 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 student {
   char name[MAX_LEN + 1];
   int id;
   void *data;
} Student;
```

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

```
typedef struct list {
   void (*free_func) (void *);
   Node *head;
} List;
```

For this problem:

- **You can assume memory allocations are always successful.**
- The driver relies on several functions (e.g., print_list) 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.

1. Implement the function **create_list** that has the prototype shown below. The function creates and returns a new dynamically-allocated **List** structure with a single student. The function will create a new **Node** structure for the student, and will assign the **new_student** pointer parameter to the **student** field of the node. The **free_func** field of the **List** structure will be initialized with the corresponding **free_func** parameter. Make sure you initialize any other field as needed. You can assume the **new_student** parameter is NOT null; the **free_func** parameter can be NULL.

```
List *create_list(void (*free_func) (void *), Student * new_student) {

        List *list = (List*)malloc(sizeof(List));
        Node *new_node = (Node*)malloc(sizeof(Node));

        new_node->next = NULL;
        new_node->student = new_student;

        list->head = new_node;
        list->free_func = free_func;

        return list;
```

GraceLoginId:

2. Implement the function **add_before** that has the prototype shown below. The function adds to the list a new student before the student whose name is associated with the **name** parameter. The function will create a **Node** structure and will initialize the **student** field with the **new_student** pointer. Make sure you initialize any other fields as needed. The function returns 0 if the no student with the specified parameter **name** is found; 1 otherwise. You can assume all parameters are not NULL.

```
int add_before(List * list, const char *name, Student * new_student) {

    Node *curr = list->head, *prev = NULL, *new_node;

    while(strcmp(curr->student->name, name) != 0){
        if(curr != NULL){
            prev = curr;
            curr = curr->next;
        } else{
            return 0;
        }
    }

    new_node = (Node*)malloc(sizeof(Node));
    new_node->student = new_student;

    if(prev == NULL){
        new_node->next = curr;
        list->head = new_node;
    } else {
        prev->next = new_node;
        new_node->next = curr;
    }

    return 1;
```

GraceLoginId:

3. Implement the function **remove_head** function that has the prototype shown below. The function removes the first element from the list. If the **free_func** field is not NULL, the **free_func** function will be called on the **data** field of the **Student** structure; otherwise no call will take place. The function must return the memory associated with the node, but it most not call free on the **student** pointer. If after removing the first element, the list is empty, the function will also return any dynamically-allocated memory associated with the **List** structure. You can assume the parameter is not NULL.

```
void remove_head(List * list) {

    Node *curr = list->head;

    list->head = list->head->next;
    if(list->free_func != NULL){
        list->free_func(curr->student->data);
    }
    free(curr);

    if(list->head == NULL){
        free(list);
    }
```

GraceLoginId:

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

Driver

```
int main() {
   Student s1 = { "Mary", 10, NULL };
   Student s2 = { "Rose", 7, NULL };
   Student s3 = { "Peter", 20, NULL };

   List *list = create_list(free, &s1);
   add_before(list, "Mary", &s2);
   add_before(list, "Rose", &s3);

   print_list(list);
   remove_head(list);
   remove_head(list);
   remove_head(list);

   return 0;
}
```

Output

```
% a.out
Peter (Id: 20)
Rose (Id: 7)
Mary (Id: 10)
%
```

GraceLoginId:

Problem #4 (Assembly)

Complete the assembly function named **replace** 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 **replace** function.
- Provide comments with your assembly code; they will be worth at least (6 pts).
- Your code must be efficient.
- The function assumes the parameters are provided via r25:24 (char *a), and r23:r22 (char target). The result is returned in r25:24.
- You may not modify the provided function and implement the modified version.
- If you need to use a register pointer, use X.
- The ascii value for '*' is 42.
- **Your solution must be recursive otherwise you will lose most of the credit for this problem.**

```
uint8_t replace(char *a, char target) {
   if (*a != '\0') {
      int cnt;
      if (*a == target) {
         *a = '*';
         cnt = 1;
      } else {
         cnt = 0;
      }

      return cnt + replace(a + 1, target);
   }

   return 0;
}
```

GraceLoginId:

```
        .global replace;
replace:
```

GraceLoginId:

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

GraceLoginId: