Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 312, Spring 2015
Aater Suleman, Instructor
Owais Khan, Chirag Sakhuja, TAs
Exam 1, March 4, 2015

Name:__________________________

Problem 1 (20 points):_______

Problem 2 (10 points):_______

Problem 3 (10 points):_______

Problem 4 (15 points):_______

Problem 5 (20 points):_______

Problem 6 (20 points):_______

Total (95 points):______________

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

I will not cheat on this exam.

____
Signature

GOOD LUCK!

Name:____

Problem 1. (20 points):

Part a. (5 points): How many bytes are required to store the following variables on an Android device that has 64 bit addresses?

```
int32_t *a;
char *b;
int32_t c;
```

Number of Bytes: $\boxed{}$

Part b. (5 points): What will the following line of code print?

```
printf("%p\n", ((int64_t *) 0x100000) + 1);
```

Output: $\boxed{}$

Part c. (5 points): Dynamically allocated memory is used when the size of the memory to be allocated is unknown at

COMPILE / RUN time (circle one answer).

Name:____

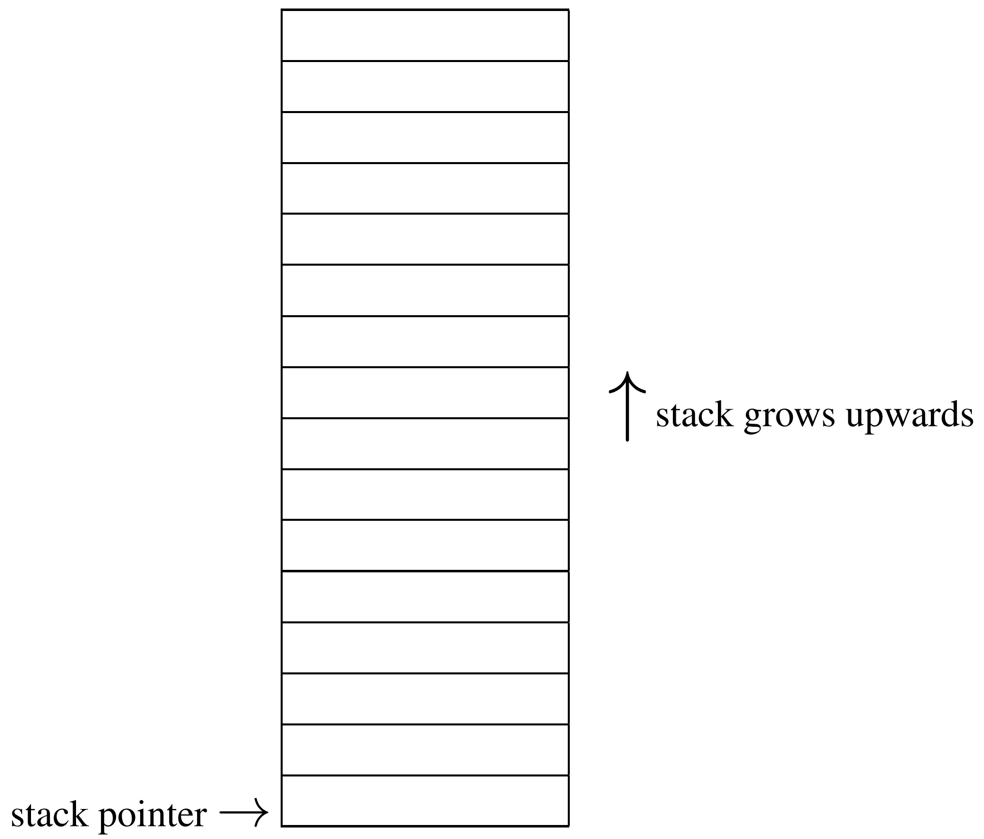
Part d. (5 points): What will the contents of the stack be once the following code has completed executing?

```
struct date {
    int32_t epoch;
};

int main() {
    int8_t a = 1;
    struct date d;
    d.epoch = 0xdeadbeef;
    char *name = (char *) 0xfffffff;
}
```

Assumptions:

- Machine is little endian (little end first)
- No compiler optimizations for alignment
- Local variables and struct elements are pushed on the stack in the order they are defined
- Pointers are 4 bytes
- Stack is initially empty (the first byte pushed on the stack will be saved at the location pointer to by the stack pointer)

Please fill in the stack entries below. Each entry corresponds to one byte. You may not need to fill in all of the entries.



Name:____

Problem 2. (10 points):

Part a. (5 points): Specify an expression in the big-Oh notation for the amount of time the following function will take in the worst case.

```
int bigoh1(int N){

    int count = 0;
    for(int i=0; i<100; i++){
        for(int j=0; j<100; j++){
            count++;
        }
    }

    return count;
}
```

O (   )

Part b. (5 points): Specify an expression in the big-Oh notation for the amount of time the following function will take in the worst case.

```
int bigoh2(int N){

    int count = 0;
    for(int i=2; i<N; i=i*i){
        count++;
    }
    return count;
}
```

O (   )

Name:____

Problem 3. (10 points):

An Aggie was hired to write a function called "calloc" that:

1. Allocates memory using malloc
2. Initializes *all* the memory locations in the allocated memory block to 0

He made a mistake while implementing the function. He wrote the following code.

```
void* calloc(int32_t size){

    unsigned char* memory = (unsigned char*) malloc(size);

    for(int32_t i=0; i<size; i++){
        *memory = 0;
        memory++;
    }

    return (void*) memory;

}

void debug1(){

    int32_t number = 1000;
    char* memory = (char*) calloc(number);
    free(memory);

}

void debug2(){

    int32_t number = 1000;
    int32_t* memory = (int32_t*) calloc(number);

    // the next few lines will be the code that uses the allocated memory
    ....
    ....

    free(memory);

}
```

Name:____

Part a. (5 points): To test the implementation, Aggie calls debug1() but the program crashes. Identify the bug in the "calloc" function and complete the code below to fix it.

Note1: You are only allowed to write one line of code in each box.
Note2: Even though there are three empty boxes, the correct solution only requires you to use one of them.
Note3: There is more than one correct solution and specifying any one of them will earn you full credit.

```
void* calloc(int32_t size){

    unsigned char* memory = (unsigned char*) malloc(size);

    ┌──────────────────────────────────────────────┐
    │                                              │
    └──────────────────────────────────────────────┘
    for(int32_t i=0; i<size; i++){

        *memory = 0;
        memory++;

        ┌──────────────────────────────────────────────┐
        │                                              │
        └──────────────────────────────────────────────┘
    }
    ┌──────────────────────────────────────────────┐
    │                                              │
    └──────────────────────────────────────────────┘
    return (void*) memory;

}
```

Part b. (5 points): After fixing the bug in the "calloc" implementation, Aggie calls debug2(). However, the Aggie does not get the correct results. Identify the bug in debug2() and complete the code below to fix it.

Write down the correct code in the blocks below.

```
void debug2(){

    int32_t number = 1000;

    ┌──────────────────────────────────────────────┐
    │                                              │
    └──────────────────────────────────────────────┘
    // next few lines contain code which use the allocated memory
    ....
    ....

    ┌──────────────────────────────────────────────┐
    │                                              │
    └──────────────────────────────────────────────┘
}
```

Name:____

Problem 4. (15 points):

Please answer the questions about the following code. You may assume that there are no compiler optimizations enabled and there is no padding on the stack.

```
int32_t *add(int32_t a, int32_t b) {
    int32_t c;
    c = a + b;
    return &c;
}

int32_t main(int32_t argc, const char *argv[]) {
    int32_t z = 0;
    int32_t *t = add(2, 1);
    z = *t;
    add(z, 1);
    printf("%d %d\n", *t, z);
    return 0;
}
```

Part a. (8 points): What is the output of the program? Output:

Please explain in less than 15 words why this output is different from the expected output of *3 3*.

Part b. (7 points): Correct the code so that it produces the expected output. Do not change the main function.
Note: For *this problem* you are not required to handle memory leaks.
Hint: The solution can be written in 3 lines of code.

```
int32_t *add(int32_t a, int32_t b) {




}
```

Name:____

Problem 5. (20 points):

In class we talked about one possible way to implement Canvas using abstract data types. In this problem, you will design the function that will be called to display the home page of Canvas when a student logs in. Below are some of the ADTs we came up. Note: Only the variables relevant to this problem are shown.

```
struct ClassType {
    ...
    /* null-terminated name of this
       class */
    char *className;

    /* number of announcements this
       class has had */
    int numAnnouncements;

    /* array of all announcements this
       this class has had */
    /* note: the size of this array is
      numAnnouncements */
    struct AnnouncementType *announce;
    ...
};

struct GradeType {
    ...
    /* class this grade was for */
    struct ClassType *class;

    /* a single grade */
    int grade;
    ...
};
```

```
struct StudentType {
    ...
    /* number of classes the student
       is enrolled in */
    int numClasses;

    /* array of all the classes the
       student is enrolled in */
    /* note: the size of this array
       is numClasses */
    struct ClassType *classes;

    /* total number of grades the
       student has received in all
       classes */
    int numGrades;

    /* array of all the grades the student
       has received */
    /* note: the size of this array
       is numGrades */
    struct GradeType *grades;
    ...
};
```

In addition to the data types, you can assume that the following functions have been implemented for you:

`uint8_t wasRecentAnnouncement(struct AnnouncementType *a)`: Given a pointer to an AnnouncementType, this function will return whether or not the announcement was made in the previous week. The return value will be a 0 if the announcement was not made in the last week, and it will be a 1 if it was made in the last week.

`void printAnnouncement(struct AnnouncementType *a)`: Given a pointer to an AnnouncementType, this function will output the date and contents of the announcement on the console in the correct format.

`int computeAverageGrade(struct GradeType *grades, struct ClassType *class)`: Given a class and the array of all the grades, this function will compute the average grade for the class. The value returned is always rounded up.

Problem continued on next page.

Name:____

Your goal is to implement the function that displays the home page of Canvas. For each class, the home page contains the name of the class, all of the announcements in that class posted within the last week (using the printAnnouncement function), and the student's average. Between each class, print the string '==========' . An example home page could look like:

```
EE312
March 02: Review session in class
Average: 92
==========
EE313
February 28: The homework assignment will be due on Monday, March 2.
Average: 95
==========
```

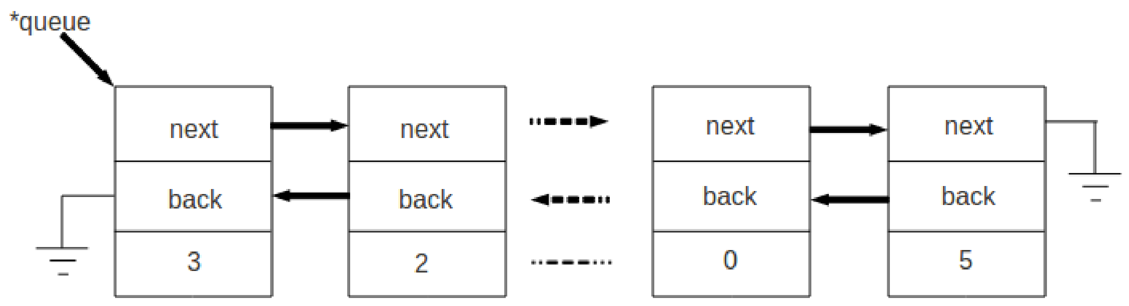
Fill in the missing code to implement the fuction printHomePage.
Note 1: You may assume that every class will have at least one recent announcement and a valid grade average.
Note 2: To print an announcement, you can simply call the printAnnouncement function.
Hint: Your code should mostly consist of calls to the functions provided. A TA wrote a solution with seven lines of code.

```
// The function will print the home page for the student passed as argument 1.
void printHomePage(struct StudentType *student) {
    for(int i = 0; i < student->numClasses; i++) {
        struct ClassType *class = student->classes[i];

        // Your code begins here




























        // Your code ends here

        printf("==========\n");
    }
}
```

Name:____

Problem 6. (20 points):

Your job is to help us complete the implementation of an abstract data type called queue. The queue supports FIFO ordering such that the first node to enter the queue must be the first node to leave the queue. Under the hood, we will implement the queue using a doubly linked list (DLL). DLL is similar to a traditional linked list (LL). In an LL, each node has a link (pointer) to the next node. In DLL, each node has two links: a link to the next node and a link to the previous node, as shown below.



Each DLL node consists of a:

1. *next pointer*: Contains either the starting address of the next node or NULL if this is the last node.
2. *back pointer*: Contains either the starting address of the previous node or NULL if this is the first node.
3. *value*: The actual value stored in the node, an integer in our example

Additionally, we maintain a queue pointer that contains either the starting address of the first node in queue or NULL if queue is empty.

The implementation consists of three functions.

1. *pushBack()*: Inserts an element at the end of the queue.
2. *popFront()*: Removes the element at the head of the queue.
3. *erase()*: Removes *all* elements in the queue which match the specified value.

Your job: Fill in the missing lines of code such that the queue behaves as specified above.

Name:____

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

typedef struct _node{

    struct _node* next;
    struct _node* back;
    int32_t value;

}node;
```

Part a. (4 points): Complete the implementation of the pushBack function by filling in the missing code

```
node* pushBack(node* queue, int32_t value){

    node* newNode = (node*) malloc(sizeof(node));
    newNode->value = value;
    newNode->next = NULL;

    if (queue == NULL){
        newNode->back = NULL;
        queue = newNode;
    }
    else{
        node* currNode = queue;
        while(currNode->next!=NULL){

            currNode = 

        }
        newNode->back  = currNode;

        currNode->next = newNode;
    }

    return queue;

}
```

Name:____

Part b. (8 points): Complete the implementation of the popFront function by filling in the missing code

```
node* popFront(node* queue,int32_t* value){

    node* firstNode = queue;

    if(queue==NULL){
        printf("Poping an empty queue.\n");
        *value = -1;
    }
    else if(firstNode->next==NULL){
        *value = firstNode->value;
        free(firstNode);
        queue = NULL;
    }
    else{
        *value = firstNode->value;

        firstNode->next->back= [                    ]

        queue = firstNode->next;

        [                    ]
    }
    return queue;
}
```

Name:____________________________________

Part c. (8 points): Complete the implementation of the erase function by filling in the missing code

```
node* erase(node* queue, int32_t value){

    node* currNode = queue;
    // no elements in queue
    if(queue == NULL){
        return queue;
    }

    // one element in queue
    if(currNode->next==NULL){
        if(currNode->value==value){
            free(currNode);
            queue = NULL;
        }
        return queue;
    }

    while(currNode->next!=NULL){

        if(currNode->value == value){
            if(currNode->back==NULL){
                currNode->next->back = NULL;
                queue = currNode->next;
                free(currNode);
                currNode = queue;
            }
            else{
                currNode->back->next = 

                currNode->next->back = 

                node* tmp = currNode->next;
                free(currNode);
                currNode = tmp;
            }
        }
        else{
            currNode = currNode->next;
        }
    }

    // check last element
    if(currNode->value== value){
        currNode->back->next = NULL;
        free(currNode);
    }

    return queue;
}
```