ZJU-UIUC Institute
Second Midterm Exam, ECE 220

Thursday 29 November 2018

Name (pinyin and Hanzi):

Student ID:

- **Be sure that your exam booklet has NINE pages.**
- **Write your name and Student ID on the first page.**
- **Do not tear the exam apart.**
- **This is a closed book exam. You may not use a calculator.**
- **You are allowed TWO handwritten A4 sheets of notes (both sides).**
- **YOU MAY NOT USE EXTRA PAPER! WRITE ON THE EXAM!**
- **Absolutely no interaction between students is allowed.**
- **Show all work, and clearly indicate any assumptions that you make.**
- **Challenge problems are marked with ***.**
- **Don't panic, and good luck!**

Problem 1	30 points	______________________
Problem 2	20 points	______________________
Problem 3	20 points	______________________
Problem 4	30 points	______________________
Total	100 points	______________________

Problem 1 (30 points): Short Answer Questions

1. **(5 points)** Consider the program below. What is the order of subroutine calls executed by the program, including only the **bar**, **foo**, and **main** functions? Give your answer as a comma-separated list.

   Answer: main , foo, bar , bar ✓

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

int32_t bar (int32_t a, int32_t b)
{
    int32_t x = a + b;
    if (0 < a) {
        printf ("%d,", a * b);
    } else {
        printf ("%d,", 0);
    }
    return x;
}

int32_t foo (int32_t* p)
{
    printf ("%d,", *p);
    *p = bar (-8, 15);
    printf ("%d,", *p);
    return 6;
}

int main ()
{
    int32_t x = 1;
    int32_t y;
    y = foo (&x);
    printf ("%d,", y);
    bar (x , y);
    return 0;
}
```

7,6

x=7

x=7

y=6

107642

y=

-3

2. **(5 points)** Write the output produced by the program above:

   Answer: 1,0,7,6,42,

3. **(5 points)** Write the output of the function below assuming that it executes on the LC-3 ISA and that the argument **val** is equal to **0x2018**.

```
void a_function (int32_t* val)
{
    printf ("0x%X\n", &val[-5]);
    printf ("0x%X\n", val + 7);
}
```

0x2018

-10

$$\begin{array}{r} 2018 \\ 14 \\ \hline 2032 \end{array}$$

F

E

E 6

First line: 0x2008E

Second line: 0x2026

Problem 1, continued:

4. **(5 points)** The **test_memory** function below crashes (the program terminates) inside the **strcpy** marked by the comment (also see the function signature and explanation below). **USING TEN WORDS OR FEWER**, explain why.

Crashes because str is still NULL after function get_memory

```
// Copies a NUL-terminated string from src to dest.  Returns dest.
char* strcpy (char* dest, const char* src);
                                                        void
void get_memory (char* p) ⇒ char* get_memory (~~char*~~ p)
{
    p = malloc (100); ⇒ return malloc (100);
}

void test_memory (void)
{
    char* str = NULL;
    get_memory (str); ⇒ str = get_memory(str);
    strcpy (str, "Hello world!"); // CRASHES INSIDE THIS CALL
    printf ("%s\n", str);
    free (str);
}
```

5. **(5 points)** Indicate how to fix the problem with the code above (mark the code directly). You may change the signatures of **get_memory** and/or **test_memory** if desired.

6. **(5 points)***** What does the function **mystery** below return? **ANSWER USING NO MORE THAN TEN WORDS.**

Answer: return the a multipule b , that is a×b mod 2^{32}

```
uint32_t
mystery (uint32_t a, uint32_t b)
{
    static uint32_t answer;

    answer = 0;
    if (0 != a) {
        mystery (a >> 1, b);
        answer <<= 1;
        if (0 != (a & 1)) {
            answer += b;
        }
    }
    return answer;
}
```

124+1

a=0 answer=0

a≠0 a右移

b·1

b·125

my(1001, b) my (500, b) my (250, b) my (125, b)

$b \cdot 2^{10} + b \cdot 2^3 + b$ $b \cdot 2^9 + b \cdot 2^2$ $b \cdot 2^8 + b \cdot 2$ $b \cdot 2^7 + b$

my(62, b) my (31, b) my (15, b)

$b \cdot 2^4$

my(7, b) my (3, b) my (1, b)

$2^2b + 2b + b$ $2b + b$ b

1024+8+1

1033

$$\begin{array}{r} 32 \\ 32 \\ \hline 64 \\ 96 \\ \hline 1024 \end{array}$$

18

+ 1032

b(

b(2^{10}+2^3+1)

my(0,b)

0

Problem 2 (20 points): Pointers and Arrays

Many graphical applications require a routine that copies the pixels from one image into another image. Write code below to copy an image into a canvas. Specifically,

- the source **image** consists of **imageHeight × imageWidth** 32-bit pixels (RGB) organized as an array of size [**imageHeight**][**imageWidth**], but passed as a **uint32_t***.
- the destination **canvas** consists of **canvasHeight × canvasWidth** 32-bit pixels (RGB) organized as an array of size [**canvasHeight**][**canvasWidth**], but passed as a **uint32_t***.
- Each pixel of **image**, (dx,dy), should be copied to the (**X**+dx,**Y**+dy) pixel of **canvas**.
- Pixels from the **image** that do not fall within the boundaries of **canvas** should be ignored.
- Note that **X** and **Y** **MAY BE NEGATIVE.**

index1,

And a few rules:

- Use at most EIGHT LINES of code (excluding curly braces and variable declarations). **Code after the first EIGHT LINES will be not graded.** It is possible to finish this problem using four lines of code.
- You must use a loop(s) in your code. Manual repetition will earn **ZERO** credit.
- No comments are needed.

```
void drawImage
    (uint32_t* image,  int32_t imageHeight,  int32_t imageWidth,
     uint32_t* canvas, int32_t canvasHeight, int32_t canvasWidth,
     int32_t X, int32_t Y) {
    int32_t   index1, index2 ;
    for (index1 =0; index1 <=imageHeight -1; index1 ++) {
          for (index2=0; index2<=imageWidth -1; index2++){                {
                 if((0 <=X+index2 <=canvasWidth -1) && (0<= Y+index1<=canvasHeight-1))
                 *(canvas+X+index2+(Y+index1)*canvesWidth) = *(image + index2 + index1*imageWidth;
                 }
          }
    }
}
```

&&

&&

Problem 3 (20 points): Testing and Debugging C Code

1. **(5 points)** Prof. Lumetta wrote the code below to compare two “signatures” consisting of two 32-bit unsigned numbers. Help him test the code by writing tests that cover all lines of the code into the table below. Provide only as many tests as are necessary to cover the code (you may not need to fill the table).

```
int32_t compare_sigs (const uint32_t* sig1, const uint32_t* sig2)
{
    int32_t i;

    for (i = 0; 2 > i; i++) {
        if (sig1[i] < sig2[i]) {
            return -1;
        } else if (sig1[i] > sig2[i]) {
            return 1;
        }
    }
    return 0;
}
```

TestNumber	sig1[0]	sig1[1]	sig2[0]	sig2[1]	return value
1	0	1	0	0	✓ 1
2	0	0	0	1	✓ -1
3	0	0	0	0	0 ✓
4					
5					

2. **(5 points)** Prof. Lumetta also considered a version using a sequential decomposition instead of a loop. The code for the alternate version is shown below.

   How many tests are necessary to cover the alternate version (in other words, what is the minimum number of tests needed to execute all statements in the code below)?

   Answer: 4 ✓

```
int32_t compare_sigs (const uint32_t* sig1, const uint32_t* sig2)
{
    if (sig1[0] < sig2[0]) {
        return -1;
    }
    if (sig1[0] > sig2[0]) {
        return 1;
    }
    if (sig1[1] < sig2[1]) {
        return -1;
    }
    return (sig1[1] > sig2[1]);
}
```

Problem 3, continued:

3. **(5 points)** Prof. Lumetta has a problem. He wrote the code below to reverse a list of **`element_t`**'s and return a pointer to the new head. The list passed cannot be empty—in other words, **`head`** is not **`NULL`**. The code passed all of Prof. Lumetta's tests. Unfortunately, he made an algorithmic error.

USING NO MORE THAN 10 WORDS, explain the error.

Answer: when list has only one element, NULL is dereferenced ✓

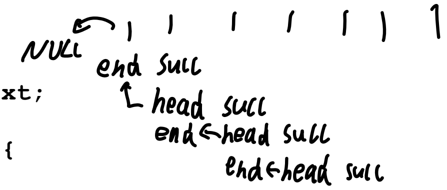
```
typedef struct element_t element_t;
struct element_t {
    int32_t    value;
    element_t* next;
};

element_t* reverse_list (element_t* head)
{
    element_t* end  = head;
    element_t* succ = head->next;

    head->next = NULL;
    head       = succ;
    succ       = head->next;

    while (NULL != succ) {
        head->next = end;
        end        = head;
        head       = succ;
        succ       = head->next;
    }

    head->next = end;
    return head;
}
```



4. **(5 points)** Prof. Lumetta has another problem. The code below is meant to remove the last element in list if the value at the head of the list is 42. The code uses the **`reverse_list`** function shown above. Unfortunately, Prof. Lumetta made a semantic error.

USING NO MORE THAN 15 WORDS, explain the error.

Answer: The address where pointer head points can't be used after freed. ✓

```
   // head is a local variable pointing to a list of element_t structures

if (42 == head->value) {
    head = reverse_list (head);      // turn the list around
    free (head);                     // free the head
    head = head->next;               // remove the old head
    head = reverse_list (head);      // and reverse the remainder
}
```

Problem 4 (30 points): The Joseph Problem

It's time for another "game" with Professor Lumetta!

This game is called the Joseph problem. N people are standing in a circle playing a game. Counting begins at a specified point in the circle and proceeds around the circle in a specified direction. After a specified number of people have been counted, the next person is out of the game. The procedure is repeated with the remaining people, starting with the person after the person who was just eliminated, going in the same direction, and counting the same number of people. When only one person remains, that person wins the game.

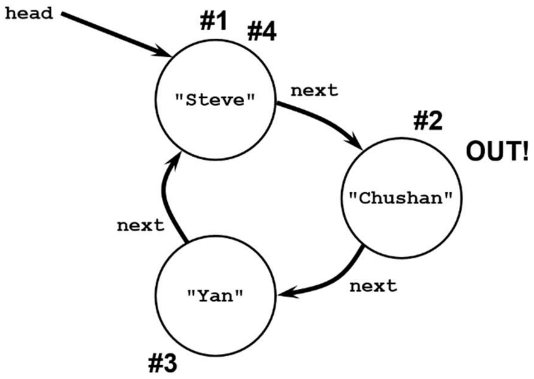
You must implement the game in C using the structure shown below. The **`next`** field is used to create a cyclic, singly-linked list. **THERE IS NO SENTINEL**.

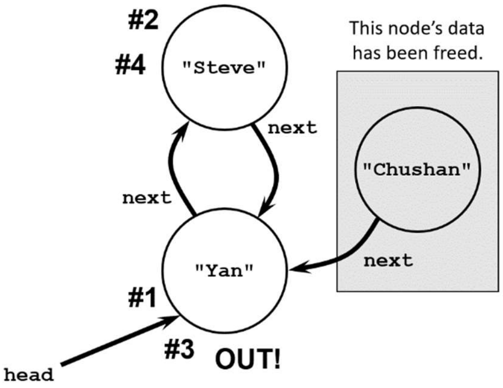
Sentinel

```
typedef struct node JosephNode;
struct node {
    char*       name; // player's name, a dynamically allocated string
    JosephNode* next; // next player in the circle
};
```

Note that both the **`JosephNode`** and the **`name`** field are dynamically allocated.

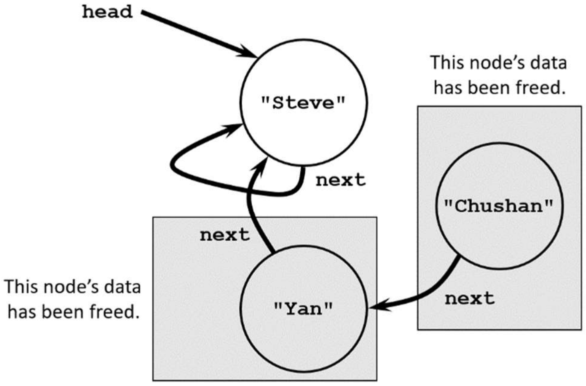
Here's an example using **`JosephNode`**s. The names do not reflect real people. The initial circle consists of three people: Steve, Chushan, and Yan. In the example, the count required is 4, and Steve is the first person to count (the **`head`** of the list). Unfortunately, Chushan is out in the first round! His node and name are freed, and Steve's **`next`** field is pointed to Yan. Yan becomes the new **`head`**.





In the second round of the game, as shown to the left, the count goes back and forth until, finally, Yan is out of the game. Yan's node and name are freed, and Steve's **`next`** field is pointed to Steve, At that point, only Steve remains, as shown below. Steve has won the game! Lucky Steve!

The final list, which includes only the winning player (Steve in this case), is shown to the right.



Problem 4, continued:

Prof. Lumetta needs your help to implement the game. You must write two C functions: one to initialize a new node, and a second to play one round of the game. Here again is the structure:

```
typedef struct node JosephNode;
struct node {
    char*       name; // player's name, a dynamically allocated string
    JosephNode* next; // next player in the circle
};
```

1. **(15 points)** Write the `JN_create` function to dynamically allocate a node and any necessary data and to copy the new node's name (the `N` parameter) into the `name` field. The function should return `NULL` on failure, or a pointer to the newly allocated `JosephNode` on success. Note that you need not initialize the `next` field. Be sure to free any allocated data if the function fails. You will need the `strlen` and `strcpy` routines from the standard C library, as well as `malloc`. Details are below.

   Only the FIRST 12 LINES of code will be graded (not counting curly braces, nor variable declarations). Only SEVEN lines are necessary to solve the problem.

```
// Copies a NUL-terminated string from src to dest.  Returns dest.
char* strcpy (char* dest, const char* src);

// Returns length of string str in bytes, not counting NUL.
size_t strlen (const char* str);

// Returns a pointer to new memory of size bytes, or NULL on failure.
void* malloc (size_t size);

JosephNode* JN_create (const char* N)
{
        JosephNode* node;
        void* result;
        node = malloc(sizeof(*node));
        result = malloc(strlen(N)+1);
        if (result != NULL && node != NULL}
            node->name = strcpy(node->name, N));   ?
            return node;}
        if (node != NULL){
            free(node);}
        if (result != NULL){
            free (resurt);}
        return NULL;
}
```

Problem 4, continued:

Here again is the structure:

```
typedef struct node JosephNode;
struct node {
    char*       name; // player's name, a dynamically allocated string
    JosephNode* next; // next player in the circle
};
```

2. **(15 points)** Write the `JN_play_round` function to play one round of the game using the cyclic, singly-linked list starting with `head` and the elimination count `count`. The list given to you (starting with `head`) **will contain at least two nodes**, and `count` **will be at least 1**. Your function must free all dynamically allocated data associated with the eliminated player (details of `free` are below). Your function should return the new `head` of the list, the player after the one eliminated.

   Only the FIRST 12 LINES of code will be graded (not counting curly braces, nor variable declarations). Only SIX lines are necessary to solve the problem.

3

1 2 3

```
// Frees the dynamically allocated memory at ptr.
void free (void* ptr);
```

```
JosephNode* JN_play_round (JosephNode* head, int32_t count)
{
        int32_t index;
        JosephNode* targert, final;
        for (index =1; index <= count; index ++){
                head = head -> next; }
        target = head -> next;
        final = target -> next; < free (target -> name);
        free( target);
        return final;
}
```