ZJU-UIUC Institute
Final Exam, ECE 220

Wednesday 2 January 2019

Name (pinyin and Hanzi):

SOLUTION IS IN RED

Student ID:

- **Be sure that your exam booklet has 12 pages.**
- **Write your name and Student ID on the first page.**
- **Some of C's I/O routines and an LC-3 ISA guide are provided. Unlike the first midterm, Patt and Patel's Appendix A will not be available during the exam.**
- **Do not tear the exam apart other than to remove the last two reference pages.**
- **This is a closed book exam. You may <u>not</u> use a calculator.**
- **You are allowed THREE 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.**
- **Don't panic, and good luck!**

Problem 1	25 points	_____
Problem 2	15 points	_____
Problem 3	20 points	_____
Problem 4	25 points	_____
Problem 4	15 points	_____

Total	100 points	_____

Problem 1 (25 points): Short Answer Questions

1. **(5 points)** The program below was designed to print the number 5 on the LC-3 display. The program does not work. **USING TEN WORDS OR FEWER**, explain why.

```
        .ORIG x3000
        JSR A
        OUT
        BRnzp DONE
A       AND R0,R0,#0
        ADD R0,R0,#5
        JSR B
        RET
DONE    HALT
ASCII   .FILL x0030
B       LD R1,ASCII
        ADD R0,R0,R1
        RET
        .END
```

Answer: subroutine A executes JSR without saving R7—infinite loop!

2. **(5 points)** As shown below, Prof. Lumetta attempted to simplify the program from **Problem 1.1** above. His version does not work, either. **USING TEN WORDS OR FEWER**, explain why.

```
        .ORIG x3000
        LD R0,NUM5
        STI R0,DDR
        HALT
NUM5    .FILL x35     ; ASCII digit '5'
DDR     .FILL xFE06
        .END
```

Answer: does not wait for display to be ready

3. **(5 points)** Prof. Lumetta's new C++ program crashes after `main` has returned. **USING TEN WORDS OR FEWER**, explain how this behavior is possible.

Answer: crashes in destructor (for variable in static storage)

Problem 1, continued:

4. **(5 points)** Read the code and fill in the outputs below.

```
#include <stdio.h>

typedef void (*func_call)(int);

void callback1 (int a)
{
    printf ("%d ", a);
}
void callback2 (int b)
{
    printf ("%d ", 9 - b);
}

func_call callback_function;

void foo (int n)
{
    int i;
    for (i = n; i < n + 3; i++) {
        if (callback_function) {
            (*callback_function) (i);
        }
    }
    printf ("\n");
}

int main ()
{
    callback_function = &callback1;
    printf ("A: ");
    foo (1);

    callback_function = NULL;
    printf ("B: ");
    foo (3);

    callback_function = &callback2;
    printf ("C: ");
    foo (5);

    return 0;
}
```

Complete the output from the program:

A: 1 2 3

B: [leave blank]

C: 4 3 2

Problem 1, continued:

5. **(5 points)** Prof. Lumetta has developed a new type of book called a "good book," as shown below.

```
typedef struct book_t book_t;
struct book_t {
    // some stuff
    book_t* next;                   // for the library
};

typedef struct good_book_t good_book_t;
struct good_book_t {
    book_t base;
    // some other stuff
    void (*promote_book) (void);    // a function pointer for good books
};
```

And he has a file scope variable with a list of books he owns…

```
static book_t* Lumetta_library = NULL;
```

Once he has populated his library with both good books and not-so-good books (plain old **`book_t`**'s), he wants to execute the following code…

```
book_t* b;

for (b = Lumetta_library; NULL != b; b = b->next) {
    good_book_t* g = (good_book_t*)b;
    g->promote_book ();
}
```

USING FIFTEEN WORDS OR FEWER, explain to Prof. Lumetta why his code keeps crashing.

Not all books are good books!

(Not safe to cast `book_t*` to `good_book_t*`.)

Problem 2 (15 points): Pareto Dominance with Recursion

This problem is based on the following node structure:

```
typedef struct node_t Node;
struct node_t {
    int32_t X;    // metric one: smaller is better
    int32_t Y;    // metric two: smaller is better
    Node*   next;
};
```

Write a recursive function that, given a pointer to the head of a singly-linked list of dynamically-allocated nodes sorted by their X values (from smallest to largest), removes all Pareto-dominated nodes from the list and frees the removed nodes. Assume that all X values are unique.

A node is Pareto-dominated if another node in the list has smaller or equal values for both X and Y.

A solution is possible using twelve lines of code. **(actually, eleven, or fourteen with non-unique X values)**

For credit, your function must be recursive.

```
void remove_dominated (Node* head)
{
    if (NULL == head || NULL == head->next) {
        return;
    }
    // original problem had non-unique X values (blue text also necessary)
    int32_t head_dom = (head->X == head->next->X && head->Y > head->next->Y);
    if (head->Y <= head->next->Y || head_dom) {
        Node* remove = head->next;
        head->next = remove->next;
        if (head_dom) {
            head->Y = remove->Y; // Note: X value is already the same.
        }
        free (remove);
        remove_dominated (head);
    } else {
        remove_dominated (head->next);
    }
}
```

Some logic: let (X_{head}, Y_{head}) be the head and (X_{next}, Y_{next}) be the next node. Let's say that there's some node (X_{dom}, Y_{dom}) that is dominated by the head. So $X_{head} \le X_{next} \le X_{dom}$ and $Y_{head} \le Y_{dom}$. If the next node is also dominated by the head, it is removed, and we start again with the same head. So either the dominated node becomes the next node, or we find a next node that is not dominated by the head. In that case, $Y_{next} < Y_{head}$. But then the next node also dominates the dominated node, so we eventually remove the dominated node.

When X values are not unique, the head itself can be dominated, but the call signature provides no way to change the head pointer of the list. Instead, we can copy a node that dominates the head over the head and remove the node from which we copied (this strategy is generally risky, since outside pointers may reference the nodes!). Since the nodes are in sorted order, any node that dominates the head must be part of a list prefix with equal X values. The head node either dominates or is dominated by the next node until only one node remains with that X value, so the X values become unique before the recursion proceeds to parts of the list with higher X values.

Problem 3 (20 points): I/O in C

In this problem, you must write a C function that processes one file to produce a second file. The input file is specified by the argument **`fname`**. The output file must be called **`out.txt`**. Your function must read characters from the input file, remove any repeated characters (case sensitive), and write the remaining characters to the output file.

For example, if the input file contains "`aaa112234abgFFrrrR`" (no quotes), the output file must contain "`a1234abgFrR`" (again, no quotes) after your function finishes writing it.

- Declare any additional variables that you need.
- Be sure to check for any possible failures and clean up any resources used.
- Return 1 on success, or 0 on failure. (Do not print error messages.)
- Remember that I/O library information is given in the reference sheet at the back of the exam.

```
int32_t file_reduce (const char* fname)
{
    FILE* in;  // input stream
    FILE* out; // output stream
    // First, write code to prepare the streams for use.

    if (NULL == (in = fopen (fname, "r")) ||
        NULL == (out = fopen ("out.txt", "w"))) {
        if (NULL != in) {
            fclose (in);
        }
        return 0;
    }
```

```
    // Read the input file and produce the output.

    int last = EOF, char;

    while (EOF != (char = fgetc (in))) {
        if (last != char) {
            fputc (char, out);
            last = char;
        }
    }
```

```
    // Clean up and return.

    fclose (in);
    return (0 == fclose (out) ? 1 : 0);

}
```

Problem 4 (25 points): Lists and Hierarchies of Structures

Recall that in class we developed container code for cyclic, doubly-linked lists with sentinels. Later, you made use of the code in a lab.

The node structure for the list appears below, and a diagram of the structure in memory when compiled for LC-3 appears to the right (with offsets).

```
typedef struct double_list_t double_list_t;
struct double_list_t {
    double_list_t* prev;  // previous element of list
    double_list_t* next;  // next element of list
};
```

+0	prev
+1	next

1. **(10 points)** Implement the list insertion code shown below as an LC-3 assembly subroutine. The diagram to the right of the code shows the stack on entry to your subroutine.
   - Do NOT set up a stack frame.
   - Use **NO MORE THAN SEVEN INSTRUCTIONS** (not counting RET, provided for you).
   - Your code may **change only R0, R1, and R2**.
   - **Do not change R6**—the subroutine returns `void`.
   - *Hint: if you put the right values into the three registers, you need only one instruction per line of C code.*

```
void
dl_insert (double_list_t* head, double_list_t* elt)
{
    elt->next = head->next;
    elt->prev = head;
    head->next->prev = elt;
    head->next = elt;
}
```

R6 ->	head
	elt

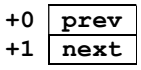
```
DL_INSERT   LDR R0,R6,#0    ; head
            LDR R1,R6,#1    ; elt
            LDR R2,R0,#1    ; head->next
            STR R2,R1,#1
            STR R0,R1,#0
            STR R1,R2,#0
            STR R1,R0,#1

            RET
```

Problem 4, continued:

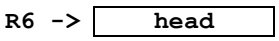
(code and diagram replicated for your convenience)

```
typedef struct double_list_t double_list_t;
struct double_list_t {
    double_list_t* prev;  // previous element of list
    double_list_t* next;  // next element of list
};
```



2. **(10 points)** Implement the code shown below to find the first element of a list as an LC-3 assembly subroutine. The diagram to the right of the code shows the stack on entry to your subroutine.
   - Do NOT set up a stack frame.
   - Use **NO MORE THAN TEN INSTRUCTIONS** (not counting RET, provided for you).
   - Your code may **change only R0, R1, R2, R3, and R6**.
   - **Be sure to push the return value on top of the stack.**

```
void*
dl_first (double_list_t* head)
{
    return (head == head->next ? NULL : head->next);
}
```




```
DL_FIRST    LDR R0,R6,#0      ; head
            LDR R1,R0,#1      ; head->next
            NOT R1,R1
            ADD R1,R1,#1
            ADD R1,R1,R0
            BRz EQUAL         ; 0 is NULL
            LDR R1,R0,#1
EQUAL       ADD R6,R6,#-1
            STR R1,R6,#0

            RET
```

3. **(5 points)** Prof. Lumetta has another issue. He has implemented a list of 3D points using the doubly-linked list code. Here is his structure definition:

```
typedef struct 3D_point_t 3D_point_t;
struct 3D_point_t {
    int32_t       x, y, z;  // coordinates of point
    double_list_t dl;       // for list of points
};
```

Here's the problem: after he fills in the coordinates for a point, he inserts the point into a list using `dl_insert`. The insertion seems to work fine, but when he looks at the coordinates of the point, they have changed! **USING TEN WORDS OR FEWER,** explain the problem.

double_list_t field must be first in 3D_point_t!

Problem 5 (15 points): C++ Call Sequencing

Read the code below.

```
#include <stdio.h>

class THING {
    int x;
public:
    THING () : x (1) {
        printf ("ONE ");
    }
    THING (int val) : x (val) {
        printf ("%d ", val);
    }
    THING& operator= (const THING& t) {
        this->x = t.x + 10;
        printf ("= %d", this->x);
        return *this;
    }
    friend THING operator+ (const THING& t, const THING& u) {
        printf ("-> ");
        return THING (t.x * u.x);
    }
};

THING* function ()
{
    printf ("line 1: ");
    THING t(1);                         // t.x is 1
    printf ("\nline 2: ");
    THING u = (t + 3) + 5;              // u.x is 15
    printf ("\nline 3: ");
    return new THING (u + (7 + 9));     // returned THING's x is 240
}
```

1. **(10 points)** Fill in the blanks below with the rest of the output produced when the subroutine `function` is called:

   line 1: `1`

   line 2: `5 3 -> 3 -> 15` `// or 3 -> 3 5 -> 15`

   line 3: `16 -> 240`

Problem 5, continued:

(code replicated for your convenience)

```
#include <stdio.h>

class THING {
    int x;
public:
    THING () : x (1) {
        printf ("ONE ");
    }
    THING (int val) : x (val) {
        printf ("%d ", val);
    }
    THING& operator= (const THING& t) {
        this->x = t.x + 10;
        printf ("= %d", this->x);
        return *this;
    }
    friend THING operator+ (const THING& t, const THING& u) {
        printf ("-> ");
        return THING (t.x * u.x);
    }
};

THING* function ()
{
    printf ("line 1: ");
    THING t(1);
    printf ("\nline 2: ");
    THING u = (t + 3) + 5;
    printf ("\nline 3: ");
    return new THING (u + (7 + 9));
}
```

2. **(5 points)** Prof. Lumetta has a dilemma. He wrote the **`main`** function below. He wants to follow the "C++ style" and declare **`THING U`** as shown, but the assignment produces no output! He has noticed that if he puts the declaration at the top of the function, as with **`THING T`**, output is produced when **`T`** is assigned a value.

```
int main ()
{
    THING  T;
    THING* ptr = function ();

    T       = *ptr;   // this line produces output
    THING U = *ptr;   // this line does not!

    delete ptr;
    return 0;
}
```

USING FIFTEEN OR FEWER WORDS, explain the problem.

Declaration of U calls (default) copy constructor [not `operator=`] which produces no output.

some of the routines from C's standard I/O library

```
// returns new stream, or NULL on failure
FILE* fopen (const char* path, const char* mode);

// returns 0 on success, or EOF on failure
int fclose (FILE* stream);
```

```
// returns char, or EOF on failure
int fgetc (FILE* stream);

// returns s, or NULL on failure
char* fgets (char* s, int size, FILE* stream);

// returns # of elements read, or 0 on failure
size_t fread (void* ptr, size_t size, size_t nmemb, FILE* stream);

// returns # of conversions, or -1 on failure (no conversions)
int fscanf (FILE* stream, const char* format, ...);

// returns # of conversions, or -1 on failure (no conversions)
int sscanf (const char* str, const char* format, ...);
```

```
// returns c, or EOF on failure
int fputc (int c, FILE* stream);

// returns value >= 0 on success, < 0 on failure
int fputs (const char* s, FILE* stream);

// returns # of elements written, or 0 on failure
size_t fwrite (const void* ptr, size_t size, size_t nmemb,
               FILE* stream);

// returns # of characters printed, or negative value on failure
int fprintf (FILE* stream, const char* format, ...);

// returns # of characters printed, or negative value on failure
int snprintf (char* str, size_t size, const char* format, ...);
```