

ECE190 Final Exam, Fall 2008

Monday 15 December

Name:

- Be sure that your exam booklet has 14 pages.
- The exam is meant to be taken apart!
- Write your name at the top of each page.
- This is a closed book exam.
- You may not use a calculator.
- You are allowed **THREE**  $8.5 \times 11$ " sheets of handwritten notes.
- Absolutely no interaction between students is allowed.
- Challenging problems are marked with \*\*\*.
- Show all of your work.
- Don't panic, and good luck!

“Some talk of an appeal unto some passion,  
Some to men's feelings, others to their reason:  
The last of these was never much the fashion,  
For reason thinks all reasoning out of season.”  
—Lord Byron, from *Don Juan*

Problem 1	18 points	_____
Problem 2	16 points	_____
Problem 3	16 points	_____
Problem 4	10 points	_____
Problem 5	20 points	_____
Problem 6	20 points	_____
Total	100 points	_____

**Problem 1** (18 points): Short Answers

Please answer concisely. If your answer requires more than a few words or a simple figure, it is probably wrong. Assume in all cases that necessary header files have been included.

**Part A** (4 points): What is the **numerical value** returned by the function below?

```
typedef enum {BLACK, YELLOW, GREEN, RED, PURPLE = 42} color_t;

color_t change_color ()
{
    color_t color = RED;

    switch (color) {
        case 2: return PURPLE;
        case 1: return GREEN;
        case 3: return YELLOW;
        default: return RED;
    }
}
```

Write your answer in this box:

**Part B** (4 points): The following line appears in an LC-3 assembly program:

```
.FILL x41
```

What could this fill value represent (circle **one answer**)?

- a) the ASCII character for 'A'
- b) the unsigned integer 65
- c) the signed integer 65
- d) all of the above (that is, it could be any of (a) through (c))
- e) none of the above, this is a HEX number

**Problem 1, continued:**

Parts C through E refer to the structure and code below.

```
typedef struct {
    int hour;
    int second;
} clock_t;

void function ()
{
    clock_t work;
    clock_t* school;
    clock_t play[24];

    school = malloc (24 * sizeof (school[0]));

    /* assume that work is initialized here */

    /* code for questions would go here... */
}
```

**Part C** (2 points): Complete the `printf` statement below to display the second field of the work clock.

```
printf ("The current second is: %d\n", _____);
```

**Part D** (4 points): Which of the following is a way to assign the value 12 to the `hour` field of element 0 of the `school` clocks (**circle all correct answers**)?

- a) `school[0].hour = 12;`                      c) `school.hour = 12;`  
b) `school[0]->hour = 12;`                      d) `school->hour = 12;`

**Part E** (4 points): Which of the following is a way to assign the value 12 to the `hour` field of element 8 of the `play` clocks (**circle all correct answers**)?

- a) `play[8].hour = 12;`                      b) `play[8]->hour = 12;`  
c) `play->hour = 12;`                      d) `play.hour = 12;`

**Problem 2** (16 points): Debugging

In this problem, you must identify errors in small snippets of code. Assume in all cases that necessary header files have been included (and are thus **not the cause of the error**).

**Part A** (4 points): The code below has **one** error. Find it and explain it concisely (in one sentence).

```
void function ()
{
    int a;
    int b;
    int size;

    a = 41;
    b = ++a;
    size = sizeof (a)++;

    return ((a - b) || (2 < size));
}
```

**Part B** (4 points): The code below has **one** error. Find it and explain it concisely (in one sentence).

```
void function ()
{
    char name[20] = "Bob";
    int age;

    printf ("Enter a number for age:");
    scanf ("%d", age);
    printf ("Hello %s! You are %d years old.\n", name, age);
}
```

**Problem 2, continued:**

**Part C** (4 points): The code below has **one** error. Find it and explain it concisely (in one sentence).

```
int n;

void function (double arg)
{
    printf ("m=%d.\n", m);
    printf ("n=%d.\n", n);
}

void caller ()
{
    double m = 3.8;

    printf ("n=%d.\n", n);
    function (m);
}
```

**Part D** (4 points): The code below was intended to print the numbers 60 through 69 inclusive, but has **two** errors. Find the errors and explain them concisely (in one sentence each).

```
void function ()
{
    int num = 60;

    while (!(num = 70))
        printf ("The number is %d\n", num);
        num++;
}
```

**Problem 3** (16 points): Recursion

The following data structure and functions are declared for a program

```
typedef struct node_t node_t;
struct node_t {
    node_t* left;
    node_t* right;
    int     value;
};

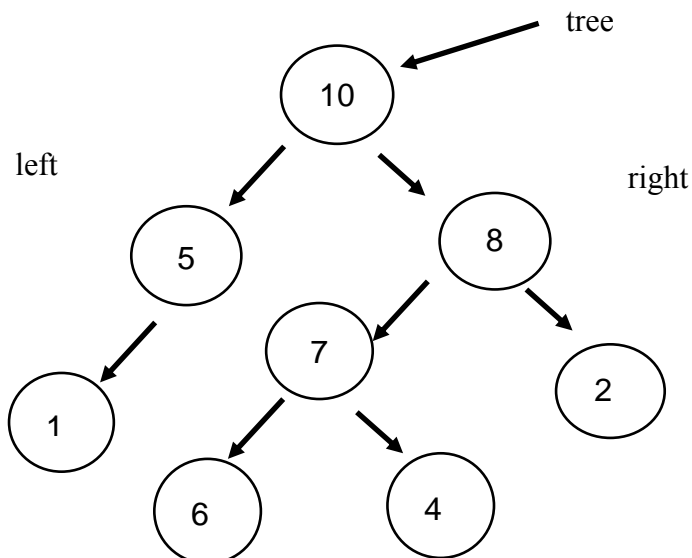
node_t* tree;

/* function body not needed for Part A */
int handle (int val, int pre, int post);

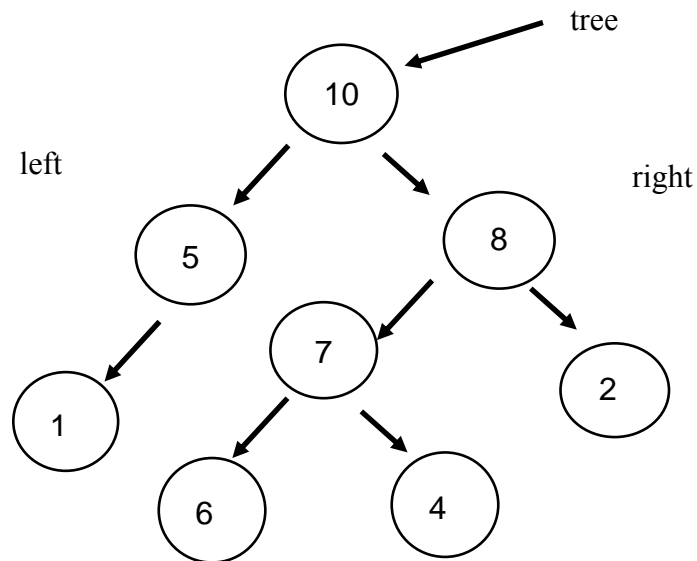
int examine (node_t* node)
{
    int left_val  = -1;
    int right_val = 1;

    if (NULL != node->right) {
        right_val = examine (node->right);
    }
    if (NULL != node->left) {
        left_val = examine (node->left);
    }
    return handle (node->value, left_val, right_val);
}
```

**Part A** (8 points): The function `examine` is called on the root of the tree shown below (node 10). List the numbers in the circles **in the order that the `handle` function** is called on each node of the tree.



Answer:

**Problem 3, continued:**

The numbers in the circles in the tree above are the `value` field for each node. Here is the code for the `handle` function:

```

int handle (int val, int pre, int post) {
    int answer;

    answer = val - pre;
    answer = answer + post;

    return answer;
}

```

**Part B** (8 points): Assume again that the `examine` function from the previous page is called on the root of the tree above (again, the root is node 10; the tree is replicated for your convenience). Fill out the table to the right with **the return value** from the `examine` function for each node in the tree. The nodes are identified by their unique `value` fields.

Node	Return Value
1	
2	
4	
5	
6	
7	
8	
10	

**\*\*\*Problem 4 (10 points):** Linked Lists and Dynamic Allocation

The node structure below is used to define a linked list of characters. Write an `interlace` function that inserts a new node with a given character before every node in a list of such nodes. For example, if the original list consists of the characters (one per node, linked into a list):

E G B D F

and you interlace the letter X, the list after calling this function should be

X E X G X B X D X F

You must write this function as an iteration (a loop) rather than using recursion: **recursion will earn no credit here**. The `head_ptr` argument points to a pointer to the first element in the original list, which the function must update as part of interleaving the list. That is, after the function executes, the value to which `head_ptr` points should have been changed to point to the new first element. The `letter_in` argument gives the letter for the new interlaced nodes. The prototype for `malloc` is provided for your convenience. You may assume that `malloc` does not fail.

```
void* malloc (size_t bytes);

typedef struct node_t node_t;
struct node_t {
    char    value;
    node_t* next;
};

void interlace (node_t** head_ptr, char letter_in)
{

```

}

**Problem 5** (20 points): I/O and Arrays

You would like to calculate your GPA. Having mastered the art of C programming in ECE190, you decide to write a C program for this purpose. After all, calculators are so “last century.”

You have a list of all your course grades, and the number of units each course was worth. Grade GPA values are 0 for F, 1 for D, etc. The information about each course is stored in the structure below:

```
typedef enum {
    GRADE_F, GRADE_D, GRADE_C, GRADE_B, GRADE_A
} grade_t;

typedef struct course_t course_t;
struct course_t {
    grade_t grade; /* letter grade received */
    int     units; /* unit credit for course */
};
```

**Part A** (10 points): The function below reads information about courses and grades from a file into an array of course structures. The function returns the number of courses read from the file (or -1 on failure). Fill in the contents of the file (bottom right) to produce the report card on the bottom left:

```
int read_grade_file (FILE* grade_report, course_t courses[])
{
    int  units = -1;
    int  n_of_c = 0;
    char letter_grade;

    while (1) {
        fscanf (grade_report, "%d %c", &units, &letter_grade);
        if (-1 == units) break;
        switch (letter_grade){
            case 'A': courses[n_of_c].grade = GRADE_A; break;
            case 'B': courses[n_of_c].grade = GRADE_B; break;
            case 'C': courses[n_of_c].grade = GRADE_C; break;
            case 'D': courses[n_of_c].grade = GRADE_D; break;
            case 'F': courses[n_of_c].grade = GRADE_F; break;
            default: printf ("Unknown grade type!\n"); return -1;
        }
        courses[n_of_c++].units = units;
    }
    return n_of_c;
}
```

Course	Grade	Units
ECE190	A	4
ECE110	A	4
ECE210	B	3

Line #	File Contents
1	
2	
3	
4	
5	

**Problem 5, continued:**

**Part B** (10 points): Write a function to calculate GPA (grade point average) given an array of `course_t` nodes and the number of courses in the array. The function must return the GPA. **For full credit, do not round the GPA off to an integer.**

The formula to calculate the GPA is:

$$\text{GPA} = \frac{\sum (\text{course grade weight} * \text{course units})}{\sum (\text{course units})}$$

where both sums are over all courses in the array. The grade names and structure definition are replicated here for your convenience. You may assume that `num_courses` (the number of courses in the array) is positive, and that course units do not sum to 0.

```
typedef enum {
    GRADE_F, GRADE_D, GRADE_C, GRADE_B, GRADE_A
} grade_t;

typedef struct course_t course_t;
struct course_t {
    grade_t grade; /* letter grade received */
    int     units; /* unit credit for course */
};

double calculate_GPA (course_t courses[], int num_courses)
{

}

}
```

**Problem 6 (20 points): LC-3 to C**

One of your TAs got excited about your MP5 and decided to extend it to support `while` loops in C. The questions on the next page refer to the code below, which was generated for a `while` loop using this extended version. The variable `num` is local to `main`'s stack frame (R5) with offset 0. `num` is initialized (before the code shown) to the value 0. Recall that R6 is the stack pointer.

```

LBL2                ; Piece 1
    LDR R0,R5,#0
    ADD R6,R6,#-1
    STR R0,R6,#0
;-----
    LD R0,LBL3      ; Piece 2
    ADD R6,R6,#-1
    STR R0,R6,#0
    BRnzp LBL4
LBL3
    .FILL #10
LBL4
;-----
    LDR R1,R6,#0    ; Piece 3
    ADD R6,R6,#1
    LDR R0,R6,#0
    ADD R6,R6,#1
    AND R2,R2,#0
    NOT R1,R1
    ADD R1,R1,#1
    ADD R0,R0,R1
    BRzp LBL5
    ADD R2,R2,#1
LBL5
    ADD R6,R6,#-1
    STR R2,R6,#0
;-----
    LDR R0,R6,#0    ; Piece 4
    ADD R6,R6,#1
    ADD R0,R0,#0
    BRnp LBL7
    LD R3,LBL6
    JMP R3
LBL6
    .FILL LBL1
LBL7
;-----
    ADD R0,R5,#0    ; Piece 5
    LDR R1,R0,#0
    ADD R2,R1,#1
    STR R2,R0,#0
    ADD R6,R6,#-1
    STR R1,R6,#0
    ADD R6,R6,#1
;-----
    LD R3,LBL8      ; Piece 6
    JMP R3
LBL8
    .FILL LBL2
LBL1

```

Name: \_\_\_\_\_

12

**Problem 6, continued:**

**Part A** (2 points): What does the code section labeled “Piece 1” do?

**Part B** (3 points): What does the code section labeled “Piece 2” do?

**Part C** (6 points): Write a **single C expression** that corresponds to the code sections 1 through 3.

**Part D** (4 points): Write a **single C statement** that corresponds to the code section labeled “Piece 5.”

**Part E** (5 points): Write the C code corresponding to the LC-3 assembly code shown on the previous page (using a few lines).

Name: \_\_\_\_\_

Use this page for scratchwork.

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

ADD	<table border="1"><tr><td>0001</td><td>DR</td><td>SR1</td><td>0</td><td>00</td><td>SR2</td></tr></table>	0001	DR	SR1	0	00	SR2	ADD DR, SR1, SR2	LD	<table border="1"><tr><td>0010</td><td>DR</td><td colspan="4">PCoffset9</td></tr></table>	0010	DR	PCoffset9				LD DR, PCoffset9
0001	DR	SR1	0	00	SR2												
0010	DR	PCoffset9															
	$DR \leftarrow SR1 + SR2, Setcc$			$DR \leftarrow M[PC + SEXT(PCoffset9)], Setcc$													
ADD	<table border="1"><tr><td>0001</td><td>DR</td><td>SR1</td><td>1</td><td colspan="2">imm5</td></tr></table>	0001	DR	SR1	1	imm5		ADD DR, SR1, imm5	LDI	<table border="1"><tr><td>1010</td><td>DR</td><td colspan="4">PCoffset9</td></tr></table>	1010	DR	PCoffset9				LDI DR, PCoffset9
0001	DR	SR1	1	imm5													
1010	DR	PCoffset9															
	$DR \leftarrow SR1 + SEXT(imm5), Setcc$			$DR \leftarrow M[M[PC + SEXT(PCoffset9)]], Setcc$													
AND	<table border="1"><tr><td>0101</td><td>DR</td><td>SR1</td><td>0</td><td>00</td><td>SR2</td></tr></table>	0101	DR	SR1	0	00	SR2	AND DR, SR1, SR2	LDR	<table border="1"><tr><td>0110</td><td>DR</td><td>BaseR</td><td colspan="3">offset6</td></tr></table>	0110	DR	BaseR	offset6			LDR DR, BaseR, offset6
0101	DR	SR1	0	00	SR2												
0110	DR	BaseR	offset6														
	$DR \leftarrow SR1 \text{ AND } SR2, Setcc$			$DR \leftarrow M[BaseR + SEXT(offset6)], Setcc$													
AND	<table border="1"><tr><td>0101</td><td>DR</td><td>SR1</td><td>1</td><td colspan="2">imm5</td></tr></table>	0101	DR	SR1	1	imm5		AND DR, SR1, imm5	LEA	<table border="1"><tr><td>1110</td><td>DR</td><td colspan="4">PCoffset9</td></tr></table>	1110	DR	PCoffset9				LEA DR, PCoffset9
0101	DR	SR1	1	imm5													
1110	DR	PCoffset9															
	$DR \leftarrow SR1 \text{ AND } SEXT(imm5), Setcc$			$DR \leftarrow PC + SEXT(PCoffset9), Setcc$													
BR	<table border="1"><tr><td>0000</td><td>n</td><td>z</td><td>p</td><td colspan="2">PCoffset9</td></tr></table>	0000	n	z	p	PCoffset9		BR{nzp} PCoffset9	NOT	<table border="1"><tr><td>1001</td><td>DR</td><td>SR</td><td colspan="3">11111</td></tr></table>	1001	DR	SR	11111			NOT DR, SR
0000	n	z	p	PCoffset9													
1001	DR	SR	11111														
	$((n \text{ AND } N) \text{ OR } (z \text{ AND } Z) \text{ OR } (p \text{ AND } P)):$ $PC \leftarrow PC + SEXT(PCoffset9)$			$DR \leftarrow \text{NOT } SR, Setcc$													
JMP	<table border="1"><tr><td>1100</td><td>000</td><td>BaseR</td><td colspan="3">000000</td></tr></table>	1100	000	BaseR	000000			JMP BaseR	ST	<table border="1"><tr><td>0011</td><td>SR</td><td colspan="4">PCoffset9</td></tr></table>	0011	SR	PCoffset9				ST SR, PCoffset9
1100	000	BaseR	000000														
0011	SR	PCoffset9															
	$PC \leftarrow BaseR$			$M[PC + SEXT(PCoffset9)] \leftarrow SR$													
JSR	<table border="1"><tr><td>0100</td><td>1</td><td colspan="4">PCoffset11</td></tr></table>	0100	1	PCoffset11				JSR PCoffset11	STI	<table border="1"><tr><td>1011</td><td>SR</td><td colspan="4">PCoffset9</td></tr></table>	1011	SR	PCoffset9				STI SR, PCoffset9
0100	1	PCoffset11															
1011	SR	PCoffset9															
	$R7 \leftarrow PC, PC \leftarrow PC + SEXT(PCoffset11)$			$M[M[PC + SEXT(PCoffset9)]] \leftarrow SR$													
TRAP	<table border="1"><tr><td>1111</td><td>0000</td><td colspan="4">trapvect8</td></tr></table>	1111	0000	trapvect8				TRAP trapvect8	STR	<table border="1"><tr><td>0111</td><td>SR</td><td>BaseR</td><td colspan="3">offset6</td></tr></table>	0111	SR	BaseR	offset6			STR SR, BaseR, offset6
1111	0000	trapvect8															
0111	SR	BaseR	offset6														
	$R7 \leftarrow PC, PC \leftarrow M[ZEXT(trapvect8)]$			$M[BaseR + SEXT(offset6)] \leftarrow SR$													