ZJU-UIUC INSTITUTE

Online Final Examination

Please read, sign and submit the honor statement on page 2 including your name, student ID, and the date. Read also the instructions on this page below before starting the exam.

Course Code: ECE220	Semester: Fall	Instructor: Pavel Loskot	
Exam Code: Paper A ■ Paper B □ Paper C □			
Exam Type: Closed-book ■ Open-book □ Partly Open-book □ Take Home □			
Exam Date: 2022/12/28	Start Time: 9am	End Time: 12pm	Duration: 3 hours
Total pages: 13		Total questions: 5	

Specific requirements and instructions to students:

- Some C's I/O routines and routines for dynamic allocation, ASCII table and LC-3 ISA guide are provided in Appendix which is available as a separate PDF file.
- This is a closed book exam, and you may not use a calculator.
- You are allowed THREE A4 sheets of handwritten or printed notes (both sides).
- Show all work, and clearly indicate any assumptions that you make.
- Challenge problems are marked with ***.
- Since this is an online exam, you have to upload your solutions back to Blackboard where they will be graded. It is recommended that handwritten solutions are scanned using a mobile phone.
- The exam ends at 12noon. You will have until 12:15pm to scan and upload your solutions. The solutions uploaded after 12:15pm will be marked as late submission and will incur some penalty points. The submission will close down at 12:30pm.

Academic Integrity

Academic integrity is essential for maintaining the quality of scholarship in the Institute and for protecting those who depend on the results of research work performed by faculty and students in the Institute. The faculty of the Zhejiang University/University of Illinois at Urbana-Champaign Institute expect all students to maintain academic integrity at all times in the classroom and the research laboratory and to conduct their academic work in accordance with the highest ethical standards of the engineering profession. Students are expected to maintain academic integrity by refraining from academic dishonesty, and by refraining from conduct which aids others in academic dishonesty, or which leads to suspicion of academic dishonesty. Violations of academic integrity will result in disciplinary actions ranging from failing grades on assignments and courses to probation, suspension or dismissal from the Institute.

Honor Statement

I have read the academic integrity statement. I promise to abide by the exam rules and regulations and agree to comport myself during the remotely administered exam in the same manner as if I were in a proctored examination room.

Please write "**I have read and will follow the Honor Statement**" on a sheet of paper and include the following information, then submit along with your answer sheet in the end.

Name:

Student ID number:

Date:

(Please go on to the next page for questions)

Problem 1 (27 points): Short Answers

Part A. (6 points) For each statement below, circle "T", if the statement is true, otherwise circle "F", if it is false. Each correct choice is worth 2 points, and no points are given for incorrect choices, or if both choices are selected.

T F Memory allocated with **`malloc()`** resides on the stack.

T F When array **`char a[10]`** is passed to a function as a parameter using the syntax **`a[]`**, the compiler copies the array contents to the stack.

T F Before an LC-3 assembly program executes (in **`lc3sim`**, for example), all registers are initialized to zero, whilst memory is not.

Part B. (5 points) Howie has a linked list created with **`node_t`** structure that contains an integer **`value`** field and a **`next`** field pointing to the next node. He wants to use GDB to examine the value of the second node in the list starting with **`head`** (a type **`node_t*`**). Suggest Howie which GDB command he should use.

_____

Part C. (5 Points) USING 15 WORDS OR FEWER, explain how to understand function **`FOO's output`**.

```
; Input: R1, a 16-bit unsigned integer
; Output: R1, 0 or 1 to indicate a property of input R1
; Registers: R2 and R7 are caller-saved
FOO
      AND   R1, R1, R1
      BRz   RT0
      ADD   R2, R1, #-1
      AND   R1, R1, R2
      BRnp  RT0
      ADD   R1, R1, #1
      RET
RT0   AND   R1, R1, #0
      RET
```

_____

Problem 1, continued:

Part D. (5 points) Believing in himself to be a good programmer, proud Keyi always lets his computer to do many boring calculations. He implemented addition of two complex numbers using C++. The class definition and the **main** function are shown below (but include headers are not shown).

```
1  class complex {
2  private:
3      int32_t real; // the real part of complex number
4      int32_t imag; // the imaginary part of complex number

5  public:
6      // two types of complex, one for int32_t, one for double
7      complex(int32_t r = 0, int32_t i = 0)    { real = r; imag = i; }
8      complex(double r = 0.0, double i = 0.0) { real = r; imag = i; }
9      complex operator+(complex const &snd) {
10         complex res;
11         res.real = this->real + snd.real;
12         res.imag = this->imag + snd.imag;
13         return res;
14     }
15     void display() { std::cout << real << " + i" << imag << std::endl; }
16 };

17 int main() {
18     complex c1(10, 24), c2(2.0, 4.8);
19     complex c3 = c1 + c2;
20     c3.display();
21 }
```

Sadly, the code above does not compile. Luckily, **EXACTLY ONE LINE CONTAINS A BUG.** Indicate which line contains the bug and explain this bug **USING TEN WORDS OR FEWER**.

The bug appears at line: _____

The bug is due to: _____

Problem 1, continued:

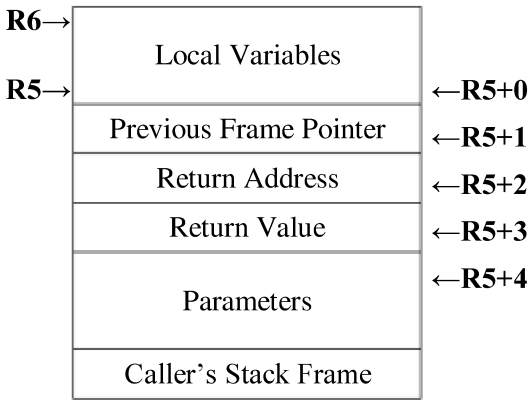
***Part E. (6 points)** The TAs on ECE220 course are going to archive the students' final grades. They wrote a program named **`grading`** having the following command-line argument format:

```
./grading <student_id> <letter_grade>
```

The **`<student_id>`** is the id of a student, which must only contain digits between 0 and 9. The **`<letter_grade>`** is the assigned letter grade between A and F. For example, "**`./grading 1 A`**" will print "Student 1 got A in ECE220!" to the terminal.

When Tianyu goes for lunch, he leaves his computer screen unlocked. If you can access his computer during his lunch, decide the input arguments to let the program print "You got an A+ in ECE220!". You can inspect the source code of the C program that Tianyu wrote below.

LC-3 Calling Convention Stack Layout




```
typedef struct grade_t {
    int16_t id;                 // student ID
    char letter[2];             // letter grade
} grade;

void buggy(int16_t index, char *s, grade *l) {
    // strcpy definition is at the end of exam
    strcpy((l + index) -> letter, s);
}

void weird() {
    printf("You got an A+ in ECE220!\n");
}

int16_t main (int16_t argc, char *argv[]) {
    grade list[10];
    int16_t i;
    for (i = 0; i < 10; i++) memset(list[i].letter, 0, 2);
    int16_t id = (int16_t) (*argv[1] - '0');
                                    // argv[1] points to input student_id
    buggy(id, argv[2], list);       // argv[2] points to input letter grade
    printf("Student %d got %s in ECE220!\n", id, list[id].letter);
    return 0;
}
```

Consider the following information to accomplish the task:

1. The program uses LC-3 memory layout (16-bit memory address), and all function calls follow the LC-3 calling conventions (the stack layout is outlined above).
2. The entry address of function **`weird()`** is at 0x3D26 (a hexadecimal number).
3. The stack frame for function **`main`** right before calling function **`buggy`** is shown below.

Please write the required arguments below:

`<student_id>` = _____

`<letter_grade>` = _____

Stack frame before call to `buggy`

Problem 2 (15 points): Linked List and Recursion

Part A. (10 points) In the 2nd midterm, we asked you to help Yingying and Howie to write function **`delete`** for a singly-linked list using the following node structure:

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

Now Howie wants you to re-implement the function **`delete`** using **RECURSION**. Complete the function **`delete`** below **USING ONLY ONE LOCAL VARIABLE.** Only the **FIRST TWELVE LINES** of code will be graded (not counting lines with only curly braces). Only **EIGHT** lines are necessary to solve this problem. *Hint: Do not forget to free the deleted data structures.*

*****ANSWERS USING MORE THAN ONE LOCAL VARIABLE WILL EARN 0 POINTS. *****

```
/* delete -- deletes the n-th element from the linked list (index-0)
 *
 * INPUTS: list_p -- points to the head of the linked list
 * n -- index of the node to be deleted (may be 0)
 * OUTPUTS: none
 * RETURN VALUE: -1 when n-th element doesn't exist, 0 otherwise
 */
int32_t delete(node_t** list_p, uint32_t n) {
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
}
```

Part B. (5 points) Yingying has two pointers **`list_p1`** and **`list_p2`** with type **`node_t**`**, pointing to the head of two singly-linked lists separately. Both link lists have sufficient and valid elements. She makes the following two calls to **`delete`**:

```
delete(list_p1, 3);
delete(list_p2, 7);
```

In the second call to **`delete`**, an error occurs inside **`free()`**. **USING 40 WORDS OR FEWER**, explain a possible reason why such error may have occurred.

Problem 3 (18 points): Function Dispatch Table and Function Dispatcher

Kaiyuan is writing a calculator in C and LC-3 with the following "Core Functions":

```
int16_t core_add(int16_t first, int16_t second);  // return first + second
int16_t core_sub(int16_t first, int16_t second);  // return first - second
int16_t core_mult(int16_t first, int16_t second); // return first * second
int16_t core_div(int16_t first, int16_t second);  // return first / second
```

Kaiyuan is fed up with calling the functions by their names, so he creates a so-called function dispatch table. In particular, since function pointers can be used to call functions, we can create an array of function pointers, and then call a specific function based on its array index instead of using the actual function name. Kaiyuan's function dispatch table (array) in LC-3 assembly has the format:

```
FDT ; function dispatch table
    .FILL core_add
    .FILL core_sub
    .FILL core_mult
    .FILL core_div
```

To make use of the function dispatch table, Kaiyuan writes a "dispatcher" function:

```
int16_t dispatcher (int16_t arg1, int16_t arg2, int16_t cmd);
```

The **`dispatcher`** uses **`cmd`** argument to select the desired "Core Function", and the arguments **`arg1`** and **`arg2`** are passed to the selected function. For example, calling **`dispatcher(2,3,0)`** invokes **`core_add`** with arguments 2 and 3 (i.e., returning 5).

The function **`dispatcher`** follows LC-3 calling conventions, but it jumps directly to the selected "Core Function," so we **DO NOT** need to set up the stack frame again. In other words, arguments to the "Core Function" do not need to be pushed onto the stack, and **R7 should not be changed** by **`dispatcher`**.

Part A. (8 points) Assuming the above description, answer the following three questions:

A.1 (4 points) Keyi contemplates to change the order of arguments **`(arg1, arg2, cmd)`** to **`(cmd, arg1, arg2)`** to call **`dispatcher`** function. **USING 40 OR FEWER WORDS,** explain whether it is a good idea or not.

A.2 (2 points) Assume that **`main`** function calls the function **`dispatcher(5,7,3)`**. Which "Core Function" is invoked? What does this "Core Function" return? Fill in the blanks below.

`dispatcher(5, 7, 3)` invokes _____, which returns _____.

A.3 (2 points) When the "Core Function" in question A.2 executes **`RET`**, to which function does it return to? **CIRCLE THE CORRECT ANSWER.**

`dispatcher` **`main`**

Problem 3, continued:

Part B. (10 points) Now Kaiyuan needs your help to finish the LC-3 implementation of **`dispatcher`**. Complete the function below. Note that you must check the **`cmd`** value, and return -1, if it is invalid. The LC3 dispatch table and **`dispatcher`** C-function are copied below for reference. Everything written beyond the lines provided will be **IGNORED**. Not all blank lines need to be filled.

```
FDT ; function dispatch table
    .FILL core_add
    .FILL core_sub
    .FILL core_mult
    .FILL core_div

  int16_t dispatcher (int16_t arg1, int16_t arg2, int16_t cmd);

  ; dispatcher - Function dispatcher to jump into the "Core Function"
  ; Input: None
  ; Output: Return 0 (on top of the stack, as in LC-3 calling convention)
  ;         if cmd is invalid.  Otherwise, returns directly
  ;         from one of the "Core Functions".
  ;
  ; R4, R5, R6, R7 follow the LC-3 calling convention.
  ; Other registers are caller-saved.
  ;

  dispatcher
      LDR R1, R5, #6      ; R1 = cmd

      ; Check if cmd is valid
       ____________________________________________________________
       ____________________________________________________________
       ____________________________________________________________
       ____________________________________________________________

      ; Get the "Core Function" pointer into R3
       ____________________________________________________________
       ____________________________________________________________
       ____________________________________________________________
       ____________________________________________________________

      JMP R3              ; Jump to the core function

      ; Jump here for invalid cmd and return 0
  INVALID_CMD
       ____________________________________________________________
       ____________________________________________________________
       ____________________________________________________________
       ____________________________________________________________
```

Problem 4 (20 points): Hierarchies of Structures and I/O

Tianyu will manage student grades stored in the following CSV (comma-separated values) file format:

```
123,78,90
456,100,88
... // more values follow
```

Each line has three fields for one student: **`ID`**, **`MP_Score`** and **`Exam_Score`**. You can assume that all **`IDs`** are unique, and that all fields are positive valid **`int32_t values`**. You can also assume that the total number of students is at least 1 and at most 200.

Part A. (8 points) Tianyu creates the following two data structures to manage the student grades:

```
typedef struct grade_t {
      int32_t id;
      int32_t final_score;
} grade_t;

typedef struct grades_t {
      int32_t num_students;     // # of students
      grade_t grade_list[200]; // student grade list
      grade_t *max;             // points to student with maximum final_score
      grade_t *min;             // points to student with minimum final_score
} grades_t;
```

Your first task is to complete the function **`read_file`** below assuming the following requirements:

1. Read student grade data from the file **`data.csv`**. *Hint: use **`fscanf()`**.*
2. Store all students' **`ID`** and **`final_score`** (sum of **`MP_Score`** and **`Exam_Score`**) into **`grade_list`** data structure (no need to sort **`final_score`** or **`ID`** values).
3. Count the total number of students and fill-in **`num_students`** field in **`grades_t`.**

The function returns 0 for success, and -1 for failure. Everything written beyond the lines provided will be **IGNORED**. Not all blank lines need to be filled.

```
// list points to a valid grades_t

int32_t read_file(grades_t *list) {
    // open the file data.csv for read
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________

    // Read the student data and fill the structure fields
    int32_t mp_score, exam_score, i = 0;
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________

    list->num_students = i;           // Fill the number of students
    _____________________________________________________________
    return 0;
}
```

Problem 4, continued:

```
typedef struct grade_t {
      int32_t id;
      int32_t final_score;
} grade_t;

typedef struct grades_t {
      int32_t num_students;    // # of students
      grade_t grade_list[200]; // student grade list
      grade_t *max;            // points to student with maximum final_score
      grade_t *min;            // points to student with minimum final_score
} grades_t;
```

Part B. (5 points) Next, complete the function **`print_max_and_min`** below that prints the maximum and the minimum **`final_score`** value among students. The code for printing has been given and you **MUST NOT** write additional printing statements. You only need to set **`max`** and **`min`** fields in **`grades_t`**. Everything written beyond the lines provided will be **IGNORED**. Not all blank lines need to be filled.

```
// list points to a valid grades_t

void print_max_and_min(grades_t *list) {

    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________
    _____________________________________________________

    printf("max: %d\n", list->max->final_score);
    printf("min: %d\n", list->min->final_score);
}
```

Problem 4, continued:

```
typedef struct grade_t {
      int32_t id;
      int32_t final_score;
} grade_t;

typedef struct grades_t {
      int32_t num_students;    // # of students
      grade_t grade_list[200]; // student grade list
      grade_t *max;            // points to student with maximum final_score
      grade_t *min;            // points to student with minimum final_score
} grades_t;
```

Part C. (7 points) Your last task is to write function **`write_file`** that outputs all students' **`ID`** and **`final_score`** from **`grades_t`** into the file **`out.txt`**. The function returns 0 for success, and -1 for failure. Everything you write beyond the lines provided will be **IGNORED**. Not all blank lines need to be filled.

The **`out.txt`** should have the following format.

```
Student_ID Final_Score
123 168
456 188
... // continue with more values
```

```
// list points to a valid grades_t

int32_t write_file(grades_t *list) {
    // Open the file out.txt for write
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________

    // Output the header to the file
    _____________________________________________________________

    // Output students' ID and final score to the file
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________
    _____________________________________________________________

    return 0;
}
```

Problem 5 (20 points): C++ Stuff

Part A. (4 points) Use the lines provided to write down the output of the following C++ code.

```
#include <iostream>
using namespace std;

class Parent {
 public:
  Parent() { cout << "Parent constructor" << endl; }
  ~Parent() { cout << "Parent destructor" << endl; }
  void disp() { cout << "disp method in Parent" << endl; }
};

class Child : public Parent {
 public:
  Child() { cout << "Child constructor" << endl; }
  ~Child() { cout << "Child destructor" << endl; }
  void disp() { cout << "disp method in Child" << endl; }
};

int main()
{
  Child child;
  Parent* obj = &child;
  obj->disp();
  child.disp();

  return 0;
}
```

_____

_____

_____

_____

_____

_____

Part B. (4 points) In no more than 20 words, explain how many 'x' will be printed out and why.

```
#include <iostream>
using namespace std;

class One {
 public: ~One() { cout << "x" << endl; }
};

void foo(One *d) { One e; *d = e; };

int main() {
  One *u = new One;
  foo(u);
  delete u;
  return 0;
}
```

Answers: _____

_____

Part C. (4 points) In no more than 20 words, explain what the following program will output and why.

```
#include <iostream>
using namespace std;

class A {
 public: void out() { cout << "A" << endl; }
};

class B : public A
{
 public: void out() { cout << "B" << endl; }
};

int main() {
  A *a;
  a = new A();
  a->out();
  a = new B();
  a->out();
  return 0;
}
```

Answers: _____

_____

Problem 5, continued:

Part D. (4 points) The following code will not compile due to access right errors. Using no more than 20 words, indicate on which lines and why these errors occur.

```
1 class BaseClass {
2  public:
3   int pubFunc() { return privMem; }
4  private:
5   int privMem;
6 };
7 class DerivedClass : public BaseClass {
8  public: void usePrivate(int i) { privMem = i; }
9 };
```

```
10 int main() {
11    BaseClass b;
12    DerivedClass d;
13    b.privMem = 1;
14    d.privMem = 1;
15 }
```

Answers: _____

_____

Part E. (4 points) Add one missing word to each blank space in order to complete the following statements.

1. Constructors are _____ functions with the same _____ as their class.
2. Constructors _____ be overloaded, but they _____ be virtual.
3. Destructor is _____ when the object goes out of _____.
4. Adding _____ in front of the argument in a function prevents accidentally _____ this argument inside the function.