

# ECE220 F22 Answer

## Problem 1

### Part

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

详解: `malloc()`分配的内存是在堆(heap)中的

2. 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. **F**

详解: 指针, 无论是字符指针还是别的指针, 在作为参数传入函数的时候, 都会退化为指针, 指向数组首个元素的地址

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

详解: 所有的寄存器肯定都初始化为0了, 但是内存中的`x3000`部分, 或者`.FILL`部分都被初始化了, 所以有一部分的memory是被初始化了的

### Part B

Howie has a liked 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.

`print head->next->value` 或者直接 `p head->next->value` 都可以

### Part C

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 RTO
    ADD R2, R1, #-1
    AND R1, R1, R2
    BRnp RTO
    ADD R1, R1, #1
    RET
RTO
    AND R1, R1, #0
    RET
```

情况1: 如果R1是0, 那么直接跳转到RTO, 返回0

情况2: 如果R1和R1-1两个数经过按位AND的操作之后不是0的话, 也返回0, 否则返回1

这体现的性质是：R1和R1-1两个数在表示为二进制的时候，16个bit上的数都应该有0，不存在某一位bit是两者都为1的情况，如果R1满足该性质，那么返回1。如果后n位数都是0，那么从右往左数第n+1个位置是1，-1之后变成011...11，在n+1之前的bits不变，所以n+1之前必须都是0。推导得到：R1必须是 $2^n$ ，其中 $n \leq 15$

答案：判断R1中的数字是否为2的n次方，其中 $n \leq 15$

## Part D

The bug appears at line: 19

The bug is due to: 2 different types data cannot be added by “+”，they need type transformation.

## Part E

暂时不会，有会的来教我，求求😊

# Problem 2

## Part A

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 the code will be graded. Only EIGHT lines are necessary to solve this problem. Hint: Do not forget to free the deleted data structures.

```
/* 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){
    if (*list_p == NULL) return -1;
    if (n == 0){
        node_t* temp = *list_p;
        *list_p = temp->next;
        free(temp);
        return 0;
    }
    return delete(&(*list_p)->next, n-1);
}
```

## Part B

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.

可能的情况是list1的第3个节点和list2的第7个节点是一个节点，调用了第一个`delete`函数之后该处内存中的地址已经被释放，第二次调用的时候就会重复释放，导致出现问题。

## Problem 3

题目大意：通过**函数调度表**(function dispatch table)和**调度器**(dispatcher)两个方法实现动态函数调用，避免直接调用函数名

1. **函数调度表 (FDT)**：一个包含四个函数指针的数组，按顺序存储 `core_add`、`core_sub`、`core_mult`、`core_div` 的地址。
2. **调度器 (dispatcher)**：接收三个参数：`arg1`、`arg2`、`cmd`（命令码）。根据 `cmd` 值选择对应的核心函数，并直接跳转执行（无需重新设置栈帧）。

### Part A

1. 不合理，根据题意，`cmd`这个参数是不需要被压入栈的，如果它位于第一个参数的位置的话，那么他原本就是最后一个被压入栈的，需要专门有一个寄存器来计数已经存进多少参数；但是如果它位于最后一个参数的话，那么它原本就是第一个被压入栈的，直接在压入的时候空出来就可以，不需要专门用一个寄存器来计算已经存入多少参数。
2. `dispatcher(5, 7, 3)` invokes `core_div`, which returns 0  
整数除法向零取整
3. 应该返回的是`main`，因为它没有重建栈帧来保存`dispatcher`函数的地址

### Part B

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
    BRn INVALID_CMD ; 检测是否<0
    ADD R2, R1, #-3
    BRp INVALID_CMD ; 检测是否>3
    ; Get the "Core Function" pointer into R3
    LEA R2, FDT ; 把R2移动到FDT标签位置
    ADD R2, R2, R1 ; 计算偏移量
    LDR R3, R2, #0
    JMP R3 ; Jump to the core function

    ; Jump here for invalid cmd and return 0

INVALID_CMD
    AND R0, R0, #0 ; 返回值为0
    ADD R6, R5, #0 ; 首先把栈顶放到第一个局部变量的位置, 也就是弹出所有的局部变量
    LDR R5, R6, #1
    LDR R7, R6, #2
    ADD R6, R6, #3 ; 把R6放到Return Value的位置
    RET

```

## Problem 4

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

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; // pointer to student with maximum final_score
    grade_t *min; // pointer 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`

```
// list points to a valid grades_t

int32_t read_file(grades_t* list){
    // open the file data.csv for read
    FILE* file = fopen("data.csv", "r");
    if (file == NULL){return 0;}
    // Read the student data and fill the structure fields
    int32_t mp_score, exam_score, i = 0;
    while (fscanf(file, "%d,%d,%d\n", &(list->grade_list[i].id), &mp_score, &exam_score) == 3){
        list->grade_list[i].final_score = mp_score + exam_score;
        i++;
    }
    list->num_students = i; // Fill the number of students
    fclose(file);
    return 0;
}
```

## Part B

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.

```
void print_max_and_min(grades_t* list){
    int32_t tmpmax = list->grade_list[0].finalscore;
    int32_t tmpmin = tmpmax;
    list->max = list->min = list->grade_list[0];
    for (int i = 0; i < list->num_students; i++){
        if (list->grade_list[i].finalscore > tmpmax){
            tmpmax = list->grade_list[i].finalscore;
            list->max = list->grade_list[i];
        } else if (list->grade_list[i].finalscore < tmpmin){
            tmpmin = list->grade_list[i].finalscore;
            list->min = list->grade_list[i];
        }
    }
    printf("max: %d\n", list->max->final_score);
    printf("min: %d\n", list->min->final_score);
}
```

## Part C

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

int32_t write_file(grades_t* list){
    // Open the file out.txt for write
    FILE* file = fopen("out.txt", "w");
    if (file == NULL){return -1;}

    // Output the header to the file
    fprintf(file, "Student_ID Final_Score\n");
    // Output students' ID and final score to the file
    for (int i = 0; i < list->num_students; i++){
        int32_t id = list->grade_list[i].id;
        int32_t score = list->grade_list[i].final_score;
        fprintf(file, "%d %d\n", id, score);
    }
    fclose(file);
}

```

## Problem 5

### Part A

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; // 先调用基类Parent的构造函数，然后调用Child自己的构造函数。
    Parent* obj = &child; // obj 是 Parent* 类型指针，指向 Child 对象。
    obj->disp(); // 由于 Parent::disp() 没有被声明为虚函数，C++ 使用静态绑定（根据指针类型而非对象实际类型决定调用的方法）。因此，调用的是 Parent 的 disp()。
    child.disp(); // 直接通过Child对象调用disp()，调用的是Child重写的版本。
    return 0; // 首先调用派生类 Child 的析构函数，然后调用基类 Parent 的析构函数（析构顺序与构造顺序相反）。
}

```

```
| }
```

## Outputs:

```
Parent constructor
Child constructor
disp method in Parent
disp method in Child
Child destructor
Parent destructor
```

## 构造与析构顺序

- **构造顺序**: 基类 → 派生类。创建派生类对象时，必须先初始化基类部分，再初始化派生类部分。
- **析构顺序**: 派生类 → 基类。析构时顺序相反，确保资源释放的正确性。

## 静态绑定 vs 动态绑定

- **静态绑定（静态多态）**: 当方法不是虚函数时，编译器根据指针或引用的类型决定调用哪个函数。示例：Parent\* obj 调用 Parent::disp()。
- **动态绑定（运行时多态）**: 如果 Parent::disp() 被声明为虚函数 (virtual void disp();)，则会根据对象的实际类型动态绑定，此时 obj->disp() 会调用 Child::disp()。

## Part B

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;
}
```

**Answer:** 会打印2个x，每行一个。第一个x是调用foo之后，局部变量e离开作用域，调用析构函数。第二个x是delete u调用析构函数。

## Part C

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;
}

```

**Answer:** 输出两个A。类A的out()不是虚函数，它是静态绑定，编译器根据指针或引用的类型决定调用哪个函数，所以即使a在后面被赋值为指向B的指针，但是它的类型仍然是A\*，所以调用的是A中的out()函数。

## Part D

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.

```

class BaseClass{
public:
    int pubFunc(){return privMem;}
private:
    int privMem;
};

class DerivedClass : public BaseClass{
public: void usePrivate(int i){privMem = i;}
};

int main(){
    BaseClass b;
    DerivedClass d;
    b.privMem = 1;
    d.privMem = 1;
}

```

**Answer:** 8, 13, 14都有问题。在主函数中不能直接访问类内的private成员。在继承的类内不能直接访问基类中的private成员

## Part E

1. Constructors are **member** functions with the same **name** as their class.
2. Constructors **can** be overloaded, but they **cannot** be virtual.
3. Destructor is **called** when the object goes out of **scope**.

4. Adding **const** in front of the argument in a function prevents accidentally **modifying** this argument inside the function.