ECE 220 F18 Answer

Problem 1

1. Subroutine A executes JSR without saving R7
2. Didn't check whether DDR is ready
3. crashes in destructor

```
Output:
A: 123
B:
C: 432
```

5. Not all books are good books, so not all elements in `Lumetta_library` is `good_book_t*`

Problem 2

```c
typedef struct node_t Node;
struct node_t{
    int32_t X;
    int32_t Y;
    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.

- 链表已按X值**从小到大** 排序
- 一个节点是帕累托劣势的，当**存在另一个节点**具有更小或相等的X和Y值
- 所有X值都是唯一的

由于链表按X升序排列，后面节点的X值必然大于当前节点的X值。因此：

- 后面的节点可能是帕累托劣势的，可能被删掉，只要验证其Y值是否>=前面的Y值

```c
void remove_dominated (Node* head){
    if (head == NULL || head->next == NULL){return;}
    remove_dominated(head->next);
    Node* current = head->next;
    Node* prev = head;
    while (current != NULL){
        if (current->Y >= head->Y){ // 假如current节点的Y值>=head节点的Y值，就把它删掉
            prev->next = current->next;
            free(current);
            current = prev->next;
        } else{
            prev = prev->next;
            current = current->next;
        }
    }
}
```

Problem 3

In this problem, you must write a C function that processes one file to produce a second file. The input fileis 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 remainingcharacters 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.)

```
int32_t file_reduce (const char* fname){
    FILE* in; // input stream
    FILE* out; // output stream
    // First, write code to prepare the streams for use
    in = fopen(fname, "r");
    out = fopen("out.txt","w");
    if (in == NULL || out == NULL){
        if (in != NULL){
            fclose(fname);
        }
        return 0;
    }
    // Read the input file and produce the output
    int last = EOF;
    while ((int character = fgetc(in)) != EOF){ // 如果输入文件读取的字符不是EOF，则...
```

```
        if (last != character){ // 如果上一个字符和下一个字符不同，则...
            fputc(character, out); // 打印该字符
            last = character; // 更新"上一个字符"
        }
    }
    // Clean upt and return
    fclose(in);
    return (0 == fclose(out)?1:0) // 如果out文件顺利关闭，则reuturn 1，反之return 0
}
```

Problem 4

```
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
};
```

1. 完成以下函数的LC3代码撰写，要求如下：
    - 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;
}
```

```
DL_INSERT ; R6一开始位于head的地址，而elt位于下一个地址
    LDR R0, R6, #0 ; R0 = head
    LDR R1, R6, #1 ; R1 = elt
    LDR R2, R0, #1 ; R2 = head->next
    STR R2, R1, #1 ; elt->next = head->next
    STR R0, R1, #0 ; elt->prev = head
    STR R1, R2, #0 ; head->next->prev = elt
    STR R1, R0, #1 ; head->next = elt
    RET
```

2. 完成以下函数的LC3代码撰写，要求如下：
    - 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 ; R6位于head的地址
    LDR R0, R6, #0 ; R0 = head
    LDR R1, R0, #1 ; R1 = head->next
    NOT R1, R1
    ADD R1, R1, #1 ; R1 = -head->next
    ADD R1, R1, R0 ; R1 = head - head->next
    BRz EQUAL
    LDR R1, R0, #1 ; R1 = head->next
EQUAL
    ADD R6, R6, #-1 ; leave space for return value
    STR R1, R6, #0
    RET
```

3. Prof. Lumetta has another issue. He has implemented a list of 3D points using the doublylinked 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.

Prof. Lumetta 的代码存在 **指针类型错误转换**，导致内存覆盖：

1. **错误传递结构体地址**：
   - 调用 `dl_insert(head, &point)` 时，错误地将整个 `3D_point_t` 结构体地址（`&point`）传入链表操作函数。
   - 正确做法应传入 `&point.dl`，即双链表节点的地址。
2. **链表操作函数的误解**：
   - `dl_insert` 期望操作 `double_list_t` 类型的节点（仅包含两个指针）。
   - 若传入`3D_point_t`的地址，函数会将该内存区域解释为`double_list_t`，导致写入 `prev` 和 `next` 指针时，覆盖了 `x` 和 `y` 的值（因为它们在内存中紧邻）。

Problem 5

```
#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));
}
```

1. Fill in the blanks below with the rest of the output produced when the subroutine `function` is called:

```
line 1: 1
line 2: 3 -> 3 5 -> 15
line 3: 16 -> 240
```

逐行分析：`THING u = (t + 3) + 5;`此处先调用构造函数`THING(int val)`并输出3，然后计算`(t+3)`通过运算符重载可得输出为`-> 3`，然后调用构造函数`THING(int val)`并输出5，最后计算`THING(3)+THING(5)`通过运算符重载可得输出为->15

2. 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.

```cpp
int main ()
{
    THING T;
    THING* ptr = function ();
    T = *ptr; // this line produces output
    THING U = *ptr; // this line does not!
    delete ptr;
    return 0;
}
```

此处因为T已经在上面定义过了，所以`T = *ptr`的=实际上是运算符重载，而`THING U = *ptr`处的=调用的是复制构造函数，如果没有则为默认复制构造函数，无输出