ECE220 F20 Answer

Problem 1

The vector header is defined as:

```
typedef struct vector{
    int length;
    node_t* head;
} vect_t;
```

Nodes in the list are defined as:

```
typedef structure node{
    double value;
    int col;
    struct node *next;
} node_t;
```

完成点乘代码：

```
double dotProc(vect_t *v1, vect_t *v2){
    // return 0 if dot product can't be performed
    if (v1 == NULL || v2 == NULL){return 0;} // 中间一个不存在
    if (v1->length != v2->length){return 0;} // 两个长度不一样
    double result = 0.0;
    node_t* h1 = v1->head;
    node_t* h2 = v2->head;
    while(h1 != NULL || h2 != NULL) {
        if (h1->col == h2->col) { // 假如两个节点所在的列数相同，则相乘相加
            result += h1->value * h2->value;
            h1 = h1->next;
            h2 = h2->next;
        } else if(h1->col < h2->col) {
            h1 = h1->next; // 如果h1的列数<h2的列数，则说明h2的那一列为0，直接把h1往后递推即可
        } else {
            h2 = h2->next; // 同理
        }
    }
    return result;
}
```

Problem 2

PART A

The returned value of `treematch()` is an enumerated, defined as:

```
typedef enum{
    SAME_ALL = 0;
    SAME_STRUCT_DIFF_VALUE = 1;
    DIFF_STRUCT = 2;
} match_t;
```

The tree node is defined as:

```
typedef structure node{
    int value;
    struct node* parent, left, right;
}
```

检验两个二叉树是完全一样，还是只有结构相同，还是完全不同

```
#include <stdio.h>
#include <stdlib.h>
#include "treematch.h"

match_t treematch(node_t* roota, node_t* root_b){
    if (roota == NULL && rootb == NULL){
        return SAME_ALL;
    }
    if (roota != NULL && rootb != NULL){
        match_t mleft = treematch(roota->left, rootb->left);
        match_t mright = treematch(roota->right, rootb->right);
        if (mleft == 2 || mright == 2){return DIFF_STRUCT;}
        else if (mleft == 1 || mright == 1){return SAME_STRUCT_DIFF_VALUE;}
        // 如果两个子树相同，则检验根节点是否相同，如果相同就是SAME_ALL
        if (roota->value == rootb->value){return SAME_ALL;}
        else{return SAME_STRUCT_DIFF_VALUE;}
    }
    return DIFF_STRUCT; // 如果一个为NULL，另外一个不是，就返回DIFF_STRUCT
}
```

PART B

1. Pre-order: 30, 15, 10, 20, 65, 50, 35, 55, 70
2. In-order: 10, 15, 20, 30, 35, 50, 55, 65, 70
3. Post-order: 10, 20, 15, 35, 55, 50, 70, 65, 30

PART C

62应该位于55的right child

Problem 3

```
#include <iostream>
using namespace std;
class Point {
private:
    double x, y;
public:
```

```
    void set_val(const double & tx, const double & ty){
        x = tx;
        y = ty;
    }
    Point():x(0),y(0){};
    Point(const Point & p):x(p.x()), y(p.y()){};
    Point(const double tx, const double ty):x(tx), y(ty){};
    ~Point(){};
    const double x() const{ return x; }
    const double y() const{ return y; }
};
class Polygon{
public:
    virtual double get_area() const = 0 ;
};
class Triangle: public Polygon {
private:
    Point point1, point2, point3;
public:
    const Point p1() const;
    const Point p2() const;
    const Point p3() const;
    Triangle();
    Triangle(const Triangle &t);
    Triangle(const Point & p1, const Point & p2, const Point & p3 );
    ~Triangle(){};
    double get_area() const;
};

Triangle::Triangle(){ // 默认构造函数
    this->point1 = Point(0, 0);
    this->point2 = Point(1, 0);
    this->point3 = Point(0, 1);
}

Triangle::Triangle(const Triangle &t){ // 复制构造函数
    this->point1 = t.point1;
    this->point2 = t.point2;
    this->point3 = t.point3;
}

Triangle::Triangle(const Point & p1, const Point & p2, const Point & p3){ // 参数化构造函数
    this->point1 = p1;
    this->point2 = p2;
    this->point3 = p3;
}

const Point Triangle::p1() const{return this->point1;}
const Point Triangle::p2() const{return this->point2;}
const Point Triangle::p3() const{return this->point3;}

double Triangle::get_area() const{
```

```
    Point v1(point2.x()-point1.x(),point2.y()-point1.y());
    Point v2(point3.x()-point1.x(), point3.y()-point1.y());
    double cross_product = (v1.x() * v2.y() - v2.x() * v1.y());
    if (cross_product < 0){cross_product = -cross_product;}
    return 0.5 * cross_product;
}
```

Problem 4

Local Variables
Caller's Frame Pointer
Return Address
Return Value
Arguments

R0	NODE
R1	SUM
R2	TEMP
R5	FRAME POINTER
R6	STACK POINTER
R7	RETURN ADDRESS

C Code:

```
typedef struct Node{
    int value;
    Node* left;
    Node* right;
}
int SumPostOrder(Node* node) {
    int sum = 0;
    if (node == NULL){
        return sum;
    }
    sum += SumPostOrder((node->left);
    sum += SumPostOrder(node->right);
    sum += node->value;
    return sum;
}
```

LC3 Code

```
SUM_POST_ORDER
    ; Callee Set-up
    ADD R6, R6, #-4 ; R6开始时在最后一个参数，也就是node的地址处，把它移动到Local Variable的第一处
    STR R7, R6, #2
    STR R5, R6, #1
    ADD R5, R6, #0 ; 现在R5和R6都位于第一个局部变量处
    AND R1, R1, #0 ; 初始化R1
    ; Check Base Case
```

```
    LDR R0, R6, #4; load NODE, R0 = node
    BRz DONE
    ; SumPostOrder(node->left)
    LDR R2, R0, #1 ; 现在R2 = node->left
    ADD R6, R6, #-1 ; update stack pointer
    STR R2, R6, #0 ; Push argument
    JSR SUM_POST_ORDER
    LDR R2, R6, #0 ; store return value in TEMP
    ADD R1, R1, R2 ; Add TEMP to SUM
    ADD R6, R6, #2 ; stack teardown

    LDR R2, R0, #0 ; 把node->value存入R2
    ADD R1, R1, R2 ; Sum += node->value
DONE
    STR R1, R6, #3 ; store return value
    ...
;  Callee Tear-down...
    RET
```

Problem 5

```
typedef int cell;
typedef struct{
    int rows;
    int cols;
    cell* cells;
    int score;
} game;
game* make_game(int rows, int cols){
    game* mygame = (game*)malloc(sizeof(game));
    mygame->cells = malloc(rows * cols * sizeof(cell));
    // rest of the function is omitted
}
```

Part A: Why is it necessary that the following function is executed after each game finishes?

```
void destroy_game(game* cur_game){
    free(cur_game->cells);
    free(cur_game);
    return NULL;
}
```

Answer: There will be no memory leak.

Part B: It is not possible to adjust the size of the `cells` array after it has been created. (True or False)

Answer; False, we can use realloc

Part C: Given the C++ classes below, answer the question in Part C. Note that the code is different from what was provided for MP12.

```
class Number{
public:
```

```
    double magnitude, phase;
Number();
    // more functions omitted
    virtual Number operator + (const Number& arg) = 0;
};
class RealNumber : private Number{
public:
    double real_component;
    RealNumber()
    RealNumber operator + (const RealNumber& arg);
    // more functions omitted
};
```

In line 12, how is the argument `arg` passed to the operator overload function?

Answer: By reference