Final Sample Answer

1. Currency Exchange

```
typedef structure product {
    char *product_name; // 占据2个byte，也就是在计算偏移量时作为1个单位
    unsigned int price; // 占据4个byte，也就是在计算偏移量时作为2个单位
    struct product* next; // 占据2个byte，也就是在计算偏移量时作为1个单位
}product; // 为了方便理解，下面统一认为三个元素各自只占用1个单位

void convert_currency(product* head_ptr, int exchange_rate){
    if (head_ptr == NULL) {return;}
    head_ptr->price *= exchange_rate;
    convert_currency(head_ptr->next, exchange_rate);
}
```

```
CONVERT_CURRENCY
    ; Callee Setup 开始时R6是栈顶，其内容是最后一个argument的地址
    ADD R6, R6, #-4 ; Reserve space on the stack for book keeping info 此时R6位于第一个局部变量，但是由于函数中没有局部变量，所以此时R6指向一个空地址
    STR R7, R6, #2 ; Store return address on the stack
    STR R5, R6, #1 ; Store caller's frame pointer on the stack
    ADD R5, R6, #0 ; Set new frame pointer 此时R5和R6都位于第一个局部变量的位置

    ; Function Logic
    LDR R0, R5, #4 ; Load argument head_ptr into R0 using R5 head_ptr是最后一个压入栈的参数，所以它在return value的下方
    BRz CALLEE_TEARDOWN ; Check if head_ptr is NULL, if so, go to callee teardown
    LDR R1, R0, #1 ; Load head_ptr->price into R1 R0是指向一个结构体的指针，price是其中的第二个参数，所以要+1
    LDR R2, R5, #5 ; Load argument exchange_rate into R2 using R5 exchange_rate是第一个压入栈的参数，所以它在return value的下面两个位置

    ; Multiply head_ptr->price by exchange_rate
    ADD R3, R1, #0 ; Set R3 to equal R1
    AND R1, R1, #0 ; Clear R1
    ; Continuously add R2 to R1 until R3 is 0
MULTIPLY_LOOP
    AND R3, R3, R3 ; Check if R3 is 0
    BRz MULTIPLY_DONE ; if R3 is 0, exit loop
```

```
    ADD R1, R1, R2 ; Add R2 to R1
    ADD R3, R3, #-1 ; Decrement R3
    BRnzp MULTIPLY_LOOP ; Continue loop
MULTIPLY_DONE
    STR R1, R0, #1 ; Store the result in R1 back into head_ptr->price R0是指向一个结构体的指针，price是其中的第二个参数，所以要+1
    ADD R6, R6, #-1 ; Move Stack pointer to reserve space for arguments 由于一共只有两个参数，而R6本来的位置就在第一个参数，所以只要-1即可
    LDR R1, R0, #2 ; Load head_ptr->next into R1 R0是指向一个结构体的指针，*next是其中的第三个参数，所以要+2
    STR R2, R5, #0 ; Put exchange_rate onto the stack as argument using R5 R5指向第一个参数，而exchange_rate就是第一个被压入栈的参数，所以不需要偏移
    STR R1, R5, #-1 ; Put head_ptr->next onto the stack as argument using R5 R5还是指向第一个参数，但是此时需要压入的是第二个参数，所以需要偏移-1
    JSR CONVERT_CURRENCY ; recuresively call convert_currency 此时R6指向最后一个被压入栈的参数，R5指向第一个被压入栈的参数，和初始状态一致，可以嵌套调用
    ADD R6, R6, #2 ; Pop return value and arguments off the stack 经过循环调用，最后会有一个head_ptr是NULL的情况，此时R6位于第一个局部变量的位置(地址中内容为空)
    ; 上面一行不是很确定偏移量具体是多少

CALLEE_TEARDOWN
    LDR R5, R6, #1 ; Restore caller's frame pointer using R6
    LDR R7, R6, #2 ; Restore return address
    ADD R6, R6, #4 ; Pop book keeping info 结束后R6位于最后一个被压入栈的argument的位置，继续后面的操作
    ; 这一行也不是很确定具体偏移量时多少
    RET            ; Return to caller
```

2. C++ Object Oriented Programming

```
class Position{
private:
    double x_,y_;
public:
    Position(){ // 非参数化构造函数（默认构造函数）
        x_ = 0.0;
        y_ = 0.0;
    }
    Position(double x, double y){ // 参数化构造函数
        x_ = x;
        y_ = y;
    }
    double getX() const {return x_;}
```

```
    double getY() const {return y_;}
    Position operator+(const Position &p) const {
        Position ret = Position(x_+p.getX(), y_+p.getY());
        return ret;
    }
    Position operator-(const Position &p) const {
        Position ret = Position(x_-p.getX(), y_-p.getY());
        return ret;
    }
};

class Grid{
public:
    virtual int getRows() const = 0;
    virtual int getCols() const = 0;
    virtual double getValue(int row, int col) const = 0;
    virtual Position getPosition(int row, int col) const = 0;
    virtual void setValue(int row, int col, double value) = 0;
    void print();
    virtual ~Grid(){}
};

class UniformGrid final : public Grid {
private:
    int rows_, cols_; // 行数和列数
    double *grid; // 用来存储2d网格中的value的1d数组
    double cellSize_; // 每一个小网格的大小
    Position p0_; // 最左上角的网格的坐标
public:
    UniformGrid(int rows, int cols, double cellSize, Position p0) {
        rows_ = rows;
        cols_ = cols;
        cellSize_ = cellSize;
        p0_ = p0;
        grid = new double[rows * cols]; // 因为所有的value都是double类型
    }
    ~UniformGrid(){delete[] grid;} // 析构函数
    int getRows() const {return rows_;}
    int getCols() const {return cols_;}
    double getValue(int row, int col) const {return grid[row * cols + col]};
    Position getPosition(int row, int col) const {
        Position p1 = Position(row, col);
        return p0_ + p1; // 利用运算符重载，在Position类中进行了定义
    }
```

```
    void setValue(int row, int col, double value){
        grid[row * cols + col] = value;
    }
};

class CurvilinearGrid final : public Grid {
private:
    int rows_, cols_;
    double *grid;
    Position *positions;
public:
    CurvilinearGrid(const int rows, const int cols, const std::list<Position> positions){
        rows_ = rows;
        cols_ = cols;
        grid = new double[rows * cols];
        for (size_t i = 0; i < rows * cols; i++){
            grid[i] = 0.0;
        }
        this->positions = new Position[rows * cols];
        int i = 0;
        for (std::list<Position>::const_iterator it = positions.begin(); it != positions.end() &&
i < rows * cols; ++it, i++){
            this->positions[i] = *it;
        }
    } // 以上为参数化构造函数
    ~CurvilinearGrid(){
        delete[] grid;
        delete[] positions;
    } // 这是析构函数
    int getRows() const {return rows_;}
    int getCols() const {return cols_;}
    double getValue(int row, int col) const {
        return grid[row * cols + col];
    }
    Position getPosition(int row, int col) const {
        return positions[row * cols + col];
    }
    void setValue(int row, int col, double value){
        grid[row * cols + col] = value;
    }
};

std::list<Position> readPositions(const char * filename){
    // Omitted...assume no error
```

```
}

double eval_function (const Position &p){
    return p.getX()*p.getX() + p.getY()*p.getY();
}

int main(){
    UniformGrid grid1(5, 5, 2.0, Position(-5,-5));
    std::list<Position> positions = readPositions("positions.csv");
    CurvilinearGrid grid2(5, 5, positions);

    std::list<Grid *> grids;
    grids.push_back(&grid1);
    grids.push_back(&grid2);

    for (Grid *grid : grids){
        for (int i = 0; i < grid->getRows(); i++){
            for (int j = 0; j < grid->getCols(); j++){
                Position p = grid->getPosition(i, j);
                double value = eval_function(p);
                grid->setvalue(i, j, value);
            }
        }
    }
    return 0;
}
```

Ben Bitdiddle的代码错误原因：`Grid* sampler = new Grid();`会编译错误，因为Grid是抽象类（包含纯虚函数），不能实例化对象。

3. Find Professor

```
typedef struct Node {
    int status; // 1(your location), 2(professor location), otherwise 0
    int visited; // 1 if the node is visited, otherwise 0
    struct Node *left;
    struct Node *right;
    struct Node * parent;
} Node;
```

```
int findProfessor(Node *node, int *distance, int limit){
    if (node == NULL || node->visited == 1 || *distance > limit){
        return 0;
    }
```

```
    // Found professor
    if (node->status == 2) {return 1;}
    // Set visited
    node->visited = 1;
    // Increase the distance
    (*distance)++;
    if (findProfessor(node->left, distance, limit)){return 1;}
    if (findProfessor(node->right, distance, limit)){return 1;}
    if (findProfessor(node->parent, distance, limit)){return 1;}
    // Undo increasing distance
    (*distance)--;
    // target is not found
    return 0;
}

int find(Node *node, int limit){
    int distance = 0;
    return findProfessor(node, &distance, limit);
}
```

4. Linked Lists

```
typedef struct node_struct{
    int data;
    struct node_struct *next;
} node;
```

```
node* rotate(node* old_head){
    node* temp = old_head;
    while (temp->next != NULL){
        temp = temp->next;
    }
    temp->next = old_head; // 把新链表的尾端变成原本的头端
    temp = old_head->next; // 把新链表的头端变成原本的头端的下一个节点
    old_head->next = NULL; // 把新链表的尾端的next设置为NULL
    return temp; // 返回新链表的头端
}

node* subK_rotate(node* head, int k){
    int i;
    node* currNode = head;
    node* nextNode;
    if (currNode == NULL || k <= 1){reutrn head;}
    i = 0;
```

```
    while (i < k && currNode->next != NULL){
        currNode = currNode->next;
        i++;
    }
    nextNode = currNode->next;
    currNode->next = NULL; // 断开前k个节点和后面的链表
    head = rotate(head); // 注意，此时我们的currNode变成了倒数第二个节点，因此我们要先把它变成最后一个节点
    if (currNode != NULL && currNode->next != NULL){
        currNode = currNode->next;
    }
    currNode->next = subK_rotate(nextNode,k);
    return head;
}
```

Part 2: 正确版本及原因
正确版本：B
原因：A无法修改头指针本身（传值），B通过二级指针修改头指针。

5. Concepts

Q1

1. **R6变化原因**：中断发生时保存上下文到栈，栈指针下移。
2. **R6恢复原因**：RTI从栈恢复寄存器，栈指针上移。

Q2

1. `num`: 数据段（全局变量）
2. `value`: 运行时栈（局部变量）
3. `ptr_arr`: 运行时栈（指针数组）
4. `*ptr_arr[0]`: 堆（malloc分配）
5. `sizeof(value)`: 8 字节（int[2]）
6. `sizeof(ptr_arr)`: 8 字节（两个32位指针）